

ENKCA-1.0

VAX 11/730 CPU
MICRO PART 1

EP-ENKCA-DL-1.0
FICHE 1 OF 2

MAY 1982
COPYRIGHT TO 1982
MADE IN USA

01010101

ENKCA-1.0

VAX 11/730 CPU
MICRO PART 1

EP-ENKCA-DL-1.0
FICHE 2 OF 2

MAY 1982
COPYRIGHT © 1982
MADE IN USA

disi20

IDENTIFICATION

Product code: ZZ-ENKCA VERSION 1.0
Product name: MICRO-DIAGNOSTIC FOR VAX-11/730 CPU MODULE
Product date: 8-MARCH-1982
Maintainer: BASE SYSTEMS DIAGNOSTIC ENGINEERING

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital or its affiliated companies.

Copyright (c) 1982 by Digital Equipment Corporation. All Rights Reserved.

The following are trademarks of Digital Equipment Corporation.

DEC
DECUS
UNIBUS

DECsystem-10
MASSBUS
VAX

DECSYSTEM-20
PDP
VMS

digital

Table of Contents

1.0	ABSTRACT	3
2.0	HARDWARE REQUIREMENTS	3
3.0	SOFTWARE REQUIREMENTS	3
4.0	PREREQUISITES	3
5.0	OPERATING INSTRUCTIONS	3
5.1	Options	3
5.2	Event Flags	4
5.3	APT Setup	4
6.0	PROGRAM FUNCTIONAL DESCRIPTION	4
6.1	Program Overview	4
6.2	Program Size	4
6.3	Program Run Times	4
6.4	Fault Detection	4
6.5	Performance During Hardware Failures	5
6.6	Program Applications	5
6.7	Test Descriptions	5
7.0	MAINTENANCE HISTORY	6

1.0 ABSTRACT

This program is a micro-diagnostic designed to test the hardware on the CPU module of the VAX-11/730 processor.

2.0 HARDWARE REQUIREMENTS

This program will run with the minimum hardware configuration of a WCS, and a CPU.

Other options (and an MCT and Array Module) may be installed without affecting the diagnostic's performance.

3.0 SOFTWARE REQUIREMENTS

This diagnostic must be run under the VAX-11/730 micro monitor (ENKAA) in a standalone environment. The micro-diagnostic is written into WCS RAM, wiping out any system micro-code that may have been resident. The micro-diagnostic monitor takes up the area where the console software is normally present.

4.0 PREREQUISITES

The WCS module and CPU hardcore are assumed to have been tested and known to be good before this diagnostic is run. The programs ENKBA through ENKBE will perform this testing.

5.0 OPERATING INSTRUCTIONS

This program may be executed by typing DI SE ENKCA after the micro-monitor has been loaded and the MIC> prompt has been printed at the terminal. Both the micro-monitor and this program are loaded from a TU58 cassette or downline loaded through the APT-RD port. See the OVERVIEW MANUAL for more information.

5.1 Options

Not applicable

5.2 Event Flags

Not applicable

5.3 APT Setup

The following describes the APT parameters needed to execute this diagnostic:
TBS

6.0 PROGRAM FUNCTIONAL DESCRIPTION

6.1 Program Overview

This program is designed to detect stuck-at faults and provide a repair tool which helps isolate the failing component. A bottom up diagnostic strategy is utilized which builds on previous tests. The tests should be run in consecutive order, to gain the best isolation of a failing component.

6.2 Program Size

This program runs in WCS RAM memory and is approximately 42K bytes long (utilizing about 14000 micro-words in WCS).

6.3 Program Run Times

This program will take approximately 12 seconds to run including initialization time and exclusive of load time.

6.4 Fault Detection

The diagnostic will report errors with the following header:

SECT TST ERR EXP REC OTHER MSK MODULE

1. SECT is the program name (ENKCA)
2. TST is the test number that failed.
3. ERR is the error number that failed.

4. EXP is the expected (correct) data.
5. REC is the received (actual) data.
6. OTHER is any other pertinent data (such as an address). This field is not always used. N/A will be printed under OTHER when not in use. The ERRORS section in the listing will describe the contents of this field when it is used.
7. MASK is the error mask used to check the result. Bits set to a 1 in this mask correspond to bits in the result that are not checked.
8. MODULE is the suspected module that caused the failure.

6.5 Performance During Hardware Failures

A power fail will cause a rebooting of the system. Other failures, such as a timeout or a WCS parity error, will print an error message and return to the MIC> prompt. The operator can then issue other commands, including a continue if desired.

6.6 Program Applications

This program can be used by field service to determine which module should be replaced and to verify that the replacement eliminated the failure. It can also be used as a repair tool to help isolate a failing component by manufacturing, field service or engineering. The program is API compatible.

Customers may use the program to verify the basic integrity of the central processor module in the system.

6.7 Test Descriptions

See the actual test in the listing for detailed descriptions and error analyses. A brief description of the logic being tested by each test can be found in the table of contents of the listing.

ZZ-ENKCA-1.0 Documentation
ENKCA MICRO-DIAGNOSTIC FOR VAX-11/730 CPU MODULE
MAINTENANCE HISTORY

G 1
Page 6
01 Mar 82

Fiche 1 Frame G1

Sequence 6

7.0 MAINTENANCE HISTORY

DATE	VERSION	DESCRIPTION
8-MAR-82	1.0	Initial release.

45	FIELD DEFINITIONS FOR MAIN MICRO WORD	VER 00.08
1051	MACRO DEFINITIONS	VER 00.02
1707	FIELD DEFINITIONS FOR DATA PATH CONTROL MICRO WORD	VER 00.01
1760	CONTROL ROM CONTENTS	VER 00.01
2509	DIAGNOSTIC MACROS AND DEFINITIONS	
3166	COMMON LS ASSIGNMENTS (TO REGISTERS)	
3192	PROGRAM SPECIFIC LS ASSIGNMENTS (LS 80-F7)	
3221	Microinstruction Formats	
3604	Tables	
3735	2901 ALU Control Description	
3838	DIAGNOSTIC MACROS	
3862	TEST POINTERS	
3999	DIAGNOSTIC POINTERS	
4089	LS DATA SECTION	
4480	PROGRAM SPECIFIC DATA SECTION (LS 80-F7)	
4856	WCS SUBROUTINES FOR CONSOLE USE	
4857	GENERATE TRANSFER DATA	
4902	DEPOSIT MCT CSR ROUTINE	
4968	EXAMINE MCT CSR ROUTINE	
5055	DEPOSIT MCT TRANSLATION BUFFER ROUTINE	
5171	EXAMINE TRANSLATION BUFFER ROUTINE	
5234	DEPOSIT UNIBUS MAP ROUTINE	
5332	EXAMINE UNIBUS MAP ROUTINE	
5395	DEPOSIT MAIN MEMORY ROUTINE	
5453	EXAMINE MAIN MEMORY ROUTINE	
5515	EXAMINE IDC ROUTINES	
5581	DEPOSIT IDC ROUTINES	
5653	SAVE WR ROUTINE	
5704	RESTORE WR ROUTINE	
5755	WCS SUBROUTINES FOR CPU USE	
5756	ERROR ROUTINE	
5923	START TRANSFER ROUTINE	
6003	General Use Subroutines	
6038	TEST 1 - Unconditional Skip Test (M8390 CPU Module)	
6350	TEST 2 - Skip, Jump On ALU Z Test (M8390 CPU Module)	
6652	TEST 3 - JSR and Stack test (including return and return+1) (M8390 CPU Module)	
7264	TEST 4 - Index Mode Into LS At Speed test (M8390 CPU Module)	
7636	TEST 5 - BIS Instruction (M8390 CPU Module)	
7945	TEST 6 - BIC Instruction (M8390 CPU Module)	
8252	TEST 7 - BIT Instruction (M8390 CPU Module)	
8556	TEST 8 - CLR LS Instruction (M8390 CPU Module)	
8855	TEST 9 - 2901 R XNOR S Function Test (M8390 CPU Module)	
9018	TEST A - Working Registers Test (M8390 CPU Module)	
9796	TEST B - 2901 R PLUS S Test (M8390 CPU Module)	
10029	TEST C - 2901 S MINUS R Test (M8390 CPU Module)	
10227	TEST D - 2901 R MINUS S Test (M8390 CPU Module)	
10449	TEST E - Q Register Test (M8390 CPU Module)	
10590	TEST F - 2901 SRC=D.Q Test (M8390 CPU Module)	
10674	TEST 10 - 2901 SRC=A.Q Test (M8390 CPU Module)	
10760	TEST 11 - 2901 SRC=0.B Test (M8390 CPU Module)	
10825	TEST 12 - 2901 DST=WRITE.B.A Test (M8390 CPU Module)	
10901	TEST 13 - 2901 RAM Shift Test (M8390 CPU Module)	
11145	TEST 14 - 2901 RAM and Q Shift Test (M8390 CPU Module)	
11582	TEST 15 - BUS D D00,D D01 from DATA TYPE CTL PAL (M8390 CPU Module).	
11741	TEST 16 - ALU N,Z Byte, Word, Long Tests (M8390 CPU Module)	

: 12132	TEST 17	- ALU V,C Byte, Word, Long Tests (M8390 CPU Module)
: 12554	TEST 18	- ALU C L, HALF CARRY L, and NOP Tests (M8390 CPU Module)
: 12937	TEST 19	- PSL Condition Code Tests (M8390 CPU Module)
: 13122	TEST 1A	- Uniqueness test on LS addressed registers (M8390 CPU Module)
: 13328	TEST 1B	- Byte and Word Write to LS (M8390 CPU Module)
: 13479	TEST 1C	- BUS D D07 and BUS D D06 Test (M8390 CPU Module)
: 13688	TEST 1D	- Branch (skip) On Condition Codes Test (M8390 CPU Module)
: 14033	TEST 1E	- Conditional Jump Tests (M8390 CPU Module)
: 14244	TEST 1F	- Skip Logic Using MUX & INCR CTL PAL Test (M8390 CPU Module)
: 14501	TEST 20	- STATE REG and skip test (M8390 CPU Module)

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03)

3-MAR-82 09:34:33

J 1
3-MAR-82

ENKCA Micro-diagnostic for DAP module

Fiche 1 Frame J1

Sequence 9
Rev 01.00

Page 3

```
:1 .TITLE "ENKCA Micro-diagnostic for DAP module Rev 01.00"
:2
:3
:4
:5 .COPYRIGHT (c) 1982 BY
:6 .DIGITAL EQUIPMENT CORPORATION, MAYNARD,
:7 .MASSACHUSETTS. ALL RIGHTS RESERVED.
:8
:9 .THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
:10 .ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE INCLUSION
:11 .OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER COPIES THEREOF
:12 .MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON. NO
:13 .TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
:14
:15 .THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE, AND
:16 .SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
:17
:18 .DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
:19 .SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
:20
:21 ++
:22 .FACILITY: VAX-11/730 MICRO-DIAGNOSTIC.
:23
:24 .ABSTRACT: This micro-diagnostic tests the hardware of the DAP module
:25 .in the VAX-11/730 processor.
:26
:27 .ENVIRONMENT: ENKAA Micro-diagnostic Monitor
:28
:29 .AUTHOR: Mike Bibeault 4-MAY-81
:30
:31 .MODIFIED BY: David Mayo 8-MAR-82 VERSION 01.00
:32 .Initial release.
:33
:34 --
:35 .nolist
:36 .list
:37 .NOCREF
:38
:39
```

:40 .NOBIN
:41 .SEQUENTIAL
:42 .RTOL
:43 .HEXADECIMAL


```

:44 .PAGE
:45 .TOC 'FIELD DEFINITIONS FOR MAIN MICRO WORD VER 00.08'
:46 .SET/UPC=<000>
:47
:48 ; THE FOLLOWING EXPRESSIONS ARE VALIDITY CHECKS FOR FIELD COMBINATIONS
:49
:50 .SET/A.VAL=<.CASE[<OPC2/>]OF[0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0]>
:51 .SET/B.VAL=<.CASE[<OPC2/>]OF[0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1]>
:52 .SET/D.VAL=<.CASE[<OPC2/>]OF[0,0,0,1,0,0,0,0,1,1,1,1,1,1,1,1]>
:53 .SET/XD.VAL=<.EQL[<OPC1/>,<OPC1/MOVE>]>
:54 .SET/CC.VAL=<.CASE[<OPC2/>]OF[0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1]>
:55 .SET/DT.VAL=<.EQL[<OPC2/>,<OPC2/MEM.REQ>]>
:56 .SET/SCTL.VAL=<.CASE[<OPC2/>]OF[0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1]>
:57 .SET/JCTL.VAL=<.CASE[<OPC2/>]OF[1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]>
:58 .SET/JUMP.VAL=<.EQL[<OPC2/>,<OPC2/JUMP>]>
:59 .SET/DECODE.VAL=<.EQL[<OPC2/>,<OPC2/DECODE>]>
:60 .SET/MISC.VAL=<.EQL[<OPC2/>,<OPC2/MISC>]>
:61 .SET/DP.VAL=<.EQL[<OPC/>,<OPC/BASIC>]>
:62
:63 THE FOLLOWING VALIDITY CHECKS ARE USE FOR THE PAGE BOUNDARY PROBLEM.
:64
:65 .SET/PAGE.PROB=<.EQL[<.AND[<7FF>,<.]>,<7FE>]>
:66 .SET/SKIP.LEGAL2=<.OR[<.EQL[<SCTL/>,<SCTL/RET.NOERR>]>,<.EQL[<SCTL/>,<SCTL/POP.USTACK>]>]>
:67
:68 .SET/SKIP.LEGAL=<.OR[<.EQL[<SCTL/>,<SCTL/NO.SKIP>]>,<.EQL[<SCTL/>,<SCTL/RETURN>]>,<.EQL[<SCTL/>,<SCTL/RETURN+1>]>,<SKIP.LEGAL2>]>
:69
:70 .SET/SKIP.PAGE.VAL=<.CASE[<PAGE.PROB>]OF[1,<SKIP.LEGAL>]>
:71 .SET/JUMP.CHECK=<.OR[<.EQL[<JCTL/>,<JCTL/JSR>]>,<.EQL[<JCTL/>,<JCTL/JSR.VALID>]>]>
:72
:73 .SET/JUMP.PAGE.VAL=<.CASE[<PAGE.PROB>]OF[1,<.XOR[1,<JUMP.CHECK>]>]>
:74
:75
:76 ; MICRO INSTRUCTION OPCODE DEFINITIONS
:77
:78 OPC/=<22>,.DEFAULT=<OPC/BASIC> ; MICRO OPCODE FIELD FOR SINGLE BIT OPCODE.
:79
:80 BASIC=1 ; OPCODE FOR 'BASIC' MICRO INSTRUCTION
:81
:82 OPC1/=<22:20> ; MICRO OPCODE FIELD FOR THREE BIT OPCODES.
:83
:84 EXTENDED=2 ; OPCODE FOR 'EXTENDED' MICRO INSTRUCTION
:85 MOVE=3 ; OPCODE FOR 'MOVE' MICRO INSTRUCTION
:86
:87 OPC2/=<22:19> ; MICRO OPCODE FIELD FOR FOUR BIT OPCODES
:88
:89 MEM.REQ=3 ; OPCODE FOR 'MEM.REQ' MICRO INSTRUCTION
:90 MISC=2 ; OPCODE FOR 'MISC' MICRO INSTRUCTION
:91 JUMP=1 ; OPCODE FOR 'JUMP' MICRO INSTRUCTION
:92 DECODE=0 ; OPCODE FOR 'DECODE' MICRO INSTRUCTION
:93
:94
:95 ; THE FOLLOWING FOUR FIELDS ARE RELATED TO ADDRESSING
:96 ; VARIOUS RAM MEMORIES WITHIN THE DATA PATH.
:97
:98 ; THE A.ADRS REFERS TO THE 2901 INTERNAL RAM 'A-PORT'

```

```

:99 : THE B.ADRS REFERS TO THE 2901 INTERNAL RAM 'B-PORT'
:100 : THE XB.ADRS REFERS TO THE 2901 INTERNAL RAM 'B-PORT'
:101 : THE D.ADRS REFERS TO THE LOWER 128 LOCATIONS OF THE LOCAL STORE RAM
:102 : THE XD.ADRS REFERS TO ALL 256 LOCATIONS OF THE LOCAL STORE RAM
:103
:104 : THE VALIDITY STATEMENTS ALLOW USAGE OF THESE FIELDS IN THE FOLLOWING MANNER
:105
:106 :     A.ADRS          EXTENDED MICRO INSTRUCTION
:107
:108 :     B.ADRS          BASIC MICRO INSTRUCTION
:109 :                     EXTENDED MICRO INSTRUCTION
:110 :                     MOVE MICRO INSTRUCTION
:111 :                     DECODE.IS MICRO INSTRUCTION
:112
:113 :     XB.ADRS          MOVE XWR MICRO INSTRUCTION
:114
:115 :     D.ADRS          BASIC MICRO INSTRUCTION
:116 :                     MEM.REQ MICRO INSTRUCTION
:117
:118 :     XD.ADRS          MOVE MICRO INSTRUCTION
:119
:120 : A.ADRS/=<10:9>,.VALIDITY=<A.VAL>
:121
:122 :     MASK=3          ; REGISTER BACKUP MASK
:123
:124 : B.ADRS/=<8:7>,.VALIDITY=<B.VAL>,.DEFAULT=0
:125
:126 :     MASK=3          ; REGISTER BACKUP MASK
:127
:128 : ; THIS ADDRESS FIELD WILL CONTAIN THE LOW TWO BITS OF THE 2901 B ADDRESS
:129 : ; THE THIRD BIT WILL BE FORCED BY HARDWARE.
:130
:131 : XB.ADRS/=<8:7>,.VALIDITY=<.EQL[<OPC1/>,<OPC1/MOVE>]>
:132
:133 :     BPC=0           ; ADDRESS OF BEGINNING PC      WR[4]
:134 :     VA=1            ; ADDRESS OF MEMORY REFERENCE WR[5]
:135 :     VAR=1           ; ADDRESS OF LAST MEMORY REFERENCE. (WR[5] IN 2901).
:136
:137 : ;D.ADRS/=<15:9>,.VALIDITY=<D.VAL>,.DEFAULT=0
:138
:139
:140 :     SAVED.2ND.REQUEST=0 ; MM SAVED STATE FOR REQUEST.
:141 :     T1=1              ; TEMP LOCATION 1
:142 :     T2=2              ; TEMP LOCATION 2
:143 :     T3=3              ; TEMP LOCATION 3
:144 :     T4=4              ; TEMP LOCATION 4
:145 :     T5=5              ; TEMP LOCATION 5
:146 :     T6=6              ; TEMP LOCATION 6
:147 :     T7=7              ; TEMP LOCATION 7
:148 :     T8=8              ; TEMP LOCATION 8
:149 :     T9=9              ; TEMP LOCATION 9
:150 :     T10=0A            ; TEMP LOCATION 10
:151 :     BACKUP.CM.PC=0A   ; COMPATIBILITY MODE KEEPS ORIGINAL PC HERE.
:152 :
:153 :     T11=0B           ; TEMP LOCATION 11
    
```



```

:154 : T12=0C : TEMP LOCATION 12
:155 : T13=0D : TEMP LOCATION 13
:156 : T14=0E : TEMP LOCATION 14
:157 : T15=0F : TEMP LOCATION 15
:158 : PC=10 : MACRO PROGRAM COUNTER
:159 : WR.BACKUP=11 : BACKUP WORKING REGISTER FOR MOV MEM.DATA TO WR[]
:160 : SAVED.VAR=12 : MM SAVED STATE FOR VAR
:161 : SAVED.DATA=13 : MM SAVED STATE FOR DATA
:162 : SAVED.PTE.ADDRESS=14 : MM SAVED STATE FOR PAGE TABLE ENTRIES ADDRESS
:163 : SAVED.PTE=15 : MM SAVED STATE FOR PAGE TABLE ENTRIES
:164 : T16=16 : TEMP LOCATION 16
:165 : T17=17 : TEMP LOCATION 17
:166 : IE.TEMP4=18 : INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION FOUR.
:167 : #FFFFFF00=19 : CONSTANT
:168 : #FFFFFF00=1A : CONSTANT
:169 : #FFFFFFFC=1B : CCNSTANT
:170 : #FFFFFFF=1C : CONSTANT
:171 :
:172 : THIS LOCATION CONTAINS THE SOFT COPY OF THE PSL. WHENEVER THE
:173 : HARDWARE VERSION OF THE PSL IS CHANGED (PSL.HW), THEN THIS
:174 : CHANGE IS ALSO REPORTED HERE BY THE MICROCODE. ALL BITS OF
:175 : THE PSL ARE REALLY LIKE THOSE IN THIS WORD EXCEPT FOR THE
:176 : CONDITION CODES WHICH MUST BE OBTAINED FROM PSL.CC
:177 :
:178 : PSL=1D : SOFT COPY OF VAX PSL
:179 : VAR=1E : VIRTUAL ADDRESS REGISTER
:180 : VAR2=1F : VIRTUAL ADDRESS REGISTER
:181 :
:182 :
:183 : #1=20 : MASK 00000001 (HEX)
:184 : #2=21 : MASK 00000002
:185 : #4=22 : MASK 00000004
:186 : #8=23 : MASK 00000008
:187 : #10=24 : MASK 00000010
:188 : #20=25 : MASK 00000020
:189 : #40=26 : MASK 00000040
:190 : #80=27 : MASK 00000080
:191 : #100=28 : MASK 00000100
:192 : #200=29 : MASK 00000200
:193 : #400=2A : MASK 00000400
:194 : #800=2B : MASK 00000800
:195 : #1000=2C : MASK 00001000
:196 : #2000=2D : MASK 00002000
:197 : #4000=2E : MASK 00004000
:198 : #8000=2F : MASK 00008000
:199 : #10000=30 : MASK 00010000
:200 : #20000=31 : MASK 00020000
:201 : #40000=32 : MASK 00040000
:202 : #80000=33 : MASK 00080000
:203 : #100000=34 : MASK 00100000
:204 : #200000=35 : MASK 00200000
:205 : #400000=36 : MASK 00400000
:206 : #800000=37 : MASK 00800000
:207 : #1000000=38 : MASK 01000000
:208 : #2000000=39 : MASK 02000000

```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) FIELD DEFINITIONS F 3-MAR-82
FIELD DEFINITIONS FOR MAIN MICRO WORD

N 1
3-MAR-82 09:34:33
ENKCA Micro-diagnostic for DAP module
VER 00.08

Fiche 1 Frame N1

Sequence 13
Rev 01.00

Page 7

:209	:	#40000000=3A	:	MASK 04000000
:210	:	#80000000=3B	:	MASK 08000000
:211	:	#100000000=3C	:	MASK 10000000
:212	:	#200000000=3D	:	MASK 20000000
:213	:	#400000000=3E	:	MASK 40000000
:214	:	#800000000=3F	:	MASK 80000000
:215	:		:	
:216	:	BIT0=20	:	MASK 00000001
:217	:	BIT1=21	:	MASK 00000002
:218	:	BIT2=22	:	MASK 00000004
:219	:	BIT3=23	:	MASK 00000008
:220	:	BIT4=24	:	MASK 00000010
:221	:	BIT5=25	:	MASK 00000020
:222	:	BIT6=26	:	MASK 00000040
:223	:	BIT7=27	:	MASK 00000080
:224	:	BIT8=28	:	MASK 00000100
:225	:	BIT9=29	:	MASK 00000200
:226	:	BIT10=2A	:	MASK 00000400
:227	:	BIT11=2B	:	MASK 00000800
:228	:	BIT12=2C	:	MASK 00001000
:229	:	BIT13=2D	:	MASK 00002000
:230	:	BIT14=2E	:	MASK 00004000
:231	:	BIT15=2F	:	MASK 00008000
:232	:	BIT16=30	:	MASK 00010000
:233	:	BIT17=31	:	MASK 00020000
:234	:	BIT18=32	:	MASK 00040000
:235	:	BIT19=33	:	MASK 00080000
:236	:	BIT20=34	:	MASK 00100000
:237	:	BIT21=35	:	MASK 00200000
:238	:	BIT22=36	:	MASK 00400000
:239	:	BIT23=37	:	MASK 00800000
:240	:	BIT24=38	:	MASK 01000000
:241	:	BIT25=39	:	MASK 02000000
:242	:	BIT26=3A	:	MASK 04000000
:243	:	BIT27=3B	:	MASK 08000000
:244	:	BIT28=3C	:	MASK 10000000
:245	:	BIT29=3D	:	MASK 20000000
:246	:	BIT30=3E	:	MASK 40000000
:247	:	BIT31=3F	:	MASK 80000000
:248	:		:	
:249	:	GPR0=40	:	GPR 0
:250	:	GPR1=41	:	GPR 1
:251	:	GPR2=42	:	GPR 2
:252	:	GPR3=43	:	GPR 3
:253	:	GPR4=44	:	GPR 4
:254	:	GPR5=45	:	GPR 5
:255	:	GPR6=46	:	GPR 6
:256	:	CM.SP=46	:	IN COMPATIBILITY THIS IS THE STACK POINTER.
:257	:	GPR7=47	:	GPR 7
:258	:	CM.PC=47	:	IN COMPATIBILITY MODE THIS IS THE PC.
:259	:	GPR8=48	:	GPR 8
:260	:	GPR9=49	:	GPR 9
:261	:	GPR10=4A	:	GPR 10
:262	:	GPR11=4B	:	GPR 11
:263	:	GPR12=4C	:	GPR 12

:264	AP=4C	: ARGUMENT POINTER
:265	GPR13=4D	: GPR 13
:266	FP=4D	: FRAME POINTER
:267	SP=4E	: GPR 14
:268	T0=4F	: TEMP LOCATION 0
:269		
:270	ZERO=50	: ZERO CONSTANT
:271	#3=51	: CONSTANT
:272	#5=52	: CONSTANT
:273	#6=53	: CONSTANT
:274	#7=54	: CONSTANT
:275	#9=55	: CONSTANT
:276	#B=56	: CONSTANT
:277	#C=57	: CONSTANT
:278	#D=58	: CONSTANT
:279	#E=59	: CONSTANT
:280	#F=5A	: CONSTANT
:281	#13=5B	: CONSTANT
:282	#1F=5C	: CONSTANT
:283	#34=5D	: CONSTANT
:284	#3B=5E	: CONSTANT
:285	#3F=5F	: CONSTANT
:286	#4B=60	: CONSTANT
:287	#66=61	: CONSTANT
:288	#A0=62	: CONSTANT
:289	#F0=63	: CONSTANT
:290	#FF=64	: CONSTANT
:291	#1FF=65	: CONSTANT
:292	#FFF=66	: CONSTANT
:293	#2001=67	: CONSTANT
:294	#3000=68	: CONSTANT
:295	#7F80=69	: CONSTANT
:296	#C000=6A	: CONSTANT
:297	#FFE0=6B	: CONSTANT
:298	#FFFF=6C	: CONSTANT
:299	#1F0000=6D	: CONSTANT
:300	#200001=6E	: CONSTANT
:301	#F27000=6F	: CONSTANT
:302	#3000000=70	: CONSTANT
:303	#41F0000=71	: CONSTANT
:304	#7000000=72	: CONSTANT
:305	#3FFFFFF00=73	: CONSTANT
:306	#7F800000=74	: CONSTANT
:307	#7FFFFFF00=75	: CONSTANT
:308	#7FFFFFF0E=76	: CONSTANT
:309	#BF800001=77	: CONSTANT
:310		
:311	GPR.OS=78	: HARDWARE INDEX TO GPRS
:312	BIT.OS(3-0)=79	: 16 LOCATION HARDWARE INDEX TO BIT MASKS
:313	BIT.OS(4-0)=7B	: 32 LOCATION HARDWARE INDEX TO BIT MASKS
:314	OS=7C	: OS REGISTER
:315	DEC.CON=7D	: DECIMAL CARRIES
:316	SIZE=7E	: SIZE REGISTER
:317	MDT= 7E	: MEMORY REFERENCE DATA SIZE
:318	INTERRUPT.VEC=7E	: HARDWARE INTERRUPT VECTOR

```

:319
:320
:321 THIS WORD IS THE HARDWARE CONDITION CODES. THE CC'S CAN BE READ
:322 OR ARE WRITTEN HERE. NO OTHER BITS IN THE PSL ARE EFFECTED.
:323
:324 PSL.CC=7F ; PSL CONDITION CODES
:325
:326 XD.ADRS/= <16:9> , .VALIDITY=<XD.VAL>
:327
:328 SAVED.2ND.REQUEST=0 ; MM SAVED STATE FOR REQUEST.
:329 T1=1 ; TEMP LOCATION 1
:330 T2=2 ; TEMP LOCATION 2
:331 T3=3 ; TEMP LOCATION 3
:332 T4=4 ; TEMP LOCATION 4
:333 T5=5 ; TEMP LOCATION 5
:334 T6=6 ; TEMP LOCATION 6
:335 T7=7 ; TEMP LOCATION 7
:336 T8=8 ; TEMP LOCATION 8
:337 T9=9 ; TEMP LOCATION 9
:338 T10=0A ; TEMP LOCATION 10
:339 BACKUP.CM.PC=0A ; COMPATIBILITY MODE KEEPS ORIGINAL PC HERE.
:340 T11=0B ; TEMP LOCATION 11
:341 T12=0C ; TEMP LOCATION 12
:342 T13=0D ; TEMP LOCATION 13
:343 T14=0E ; TEMP LOCATION 14
:344 T15=0F ; TEMP LOCATION 15
:345 PC=10 ; MACRO INSTRUCTION PC
:346 WR.BACKUP=11 ; BACKUP WORKING REGISTER FOR MOV MEM.DATA TO WR[ ].
:347 SAVED.VAR=12 ; MM SAVED VAR LOCATION
:348 SAVED.DATA=13 ; MM SAVED DATA LOCATION
:349 SAVED.PTE.ADDRESS=14 ; MM SAVED PAGE TABLE ENTRY ADDRESS
:350 SAVED.PTE=15 ; MM SAVED PAGE TABLE ENTRY
:351 T16=16 ; TEMP LOCATION 16
:352 T17=17 ; TEMP LOCATION 17
:353 IE.TEMP4=18 ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION FOUR.
:354 #FFFFFF00=19 ; CONSTANT
:355 #FFFFFF00=1A ; CONSTANT
:356 #FFFFFFFC=1B ; CONSTANT
:357 #FFFFFFF=1C ; CONSTANT
:358
:359 THIS LOCATION CONTAINS THE SOFT COPY OF THE PSL. WHENEVER THE HARDWARE
:360 VERSION OF THE PSL IS CHANGED (PSL.HW) THEN THIS CHANGE IS
:361 REPORTED HERE. ALL BITS OF THE PSL ARE REALLY LIKE THOSE HERE
:362 EXCEPT FOR THE CONDITION CODES WHICH MUST BE OBTAINED
:363 FROM PSL.CC.
:364
:365 PSL=1D ; SOFT COPY OF VAX PSL
:366 VAR=1E ; VIRTUAL ADDRESS REGISTER
:367 VAR2=1F ; VIRTUAL ADDRESS REGISTER
:368
:369 #1=20 ; MASK 00000001 (HEX)
:370 #2=21 ; MASK 00000002
:371 #4=22 ; MASK 00000004
:372 #8=23 ; MASK 00000008
:373 #10=24 ; MASK 00000010

```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) FIELD DEFINITIONS F 3-MAR-82
FIELD DEFINITIONS FOR MAIN MICRO WORD

D 2
3-MAR-82

Fiche 1 Frame D2
ENKCA Micro-diagnostic for DAP module
VER 00.08

Sequence 16
Rev 01.00

Page 10

:374	:	#20=25	:	MASK	00000020
:375	:	#40=26	:	MASK	00000040
:376	:	#80=27	:	MASK	00000080
:377	:	#100=28	:	MASK	00000100
:378	:	#200=29	:	MASK	00000200
:379	:	#400=2A	:	MASK	00000400
:380	:	#800=2B	:	MASK	00000800
:381	:	#1000=2C	:	MASK	00001000
:382	:	#2000=2D	:	MASK	00002000
:383	:	#4000=2E	:	MASK	00004000
:384	:	#8000=2F	:	MASK	00008000
:385	:	#10000=30	:	MASK	00010000
:386	:	#20000=31	:	MASK	00020000
:387	:	#40000=32	:	MASK	00040000
:388	:	#80000=33	:	MASK	00080000
:389	:	#100000=34	:	MASK	00100000
:390	:	#200000=35	:	MASK	00200000
:391	:	#400000=36	:	MASK	00400000
:392	:	#800000=37	:	MASK	00800000
:393	:	#1000000=38	:	MASK	01000000
:394	:	#2000000=39	:	MASK	02000000
:395	:	#4000000=3A	:	MASK	04000000
:396	:	#8000000=3B	:	MASK	08000000
:397	:	#10000000=3C	:	MASK	10000000
:398	:	#20000000=3D	:	MASK	20000000
:399	:	#40000000=3E	:	MASK	40000000
:400	:	#80000000=3F	:	MASK	80000000
:401	:		:		
:402	:	BIT0=20	:	MASK	00000001
:403	:	BIT1=21	:	MASK	00000002
:404	:	BIT2=22	:	MASK	00000004
:405	:	BIT3=23	:	MASK	00000008
:406	:	BIT4=24	:	MASK	00000010
:407	:	BIT5=25	:	MASK	00000020
:408	:	BIT6=26	:	MASK	00000040
:409	:	BIT7=27	:	MASK	00000080
:410	:	BIT8=28	:	MASK	00000100
:411	:	BIT9=29	:	MASK	00000200
:412	:	BIT10=2A	:	MASK	00000400
:413	:	BIT11=2B	:	MASK	00000800
:414	:	BIT12=2C	:	MASK	00001000
:415	:	BIT13=2D	:	MASK	00002000
:416	:	BIT14=2E	:	MASK	00004000
:417	:	BIT15=2F	:	MASK	00008000
:418	:	BIT16=30	:	MASK	00010000
:419	:	BIT17=31	:	MASK	00020000
:420	:	BIT18=32	:	MASK	00040000
:421	:	BIT19=33	:	MASK	00080000
:422	:	BIT20=34	:	MASK	00100000
:423	:	BIT21=35	:	MASK	00200000
:424	:	BIT22=36	:	MASK	00400000
:425	:	BIT23=37	:	MASK	00800000
:426	:	BIT24=38	:	MASK	01000000
:427	:	BIT25=39	:	MASK	02000000
:428	:	BIT26=3A	:	MASK	04000000

:429	:	BIT27=3B	:	MASK 08000000
:430	:	BIT28=3C	:	MASK 10000000
:431	:	BIT29=3D	:	MASK 20000000
:432	:	BIT30=3E	:	MASK 40000000
:433	:	BIT31=3F	:	MASK 80000000
:434	:			
:435	:	GPR0=40	:	GPR 0
:436	:	GPR1=41	:	GPR 1
:437	:	GPR2=42	:	GPR 2
:438	:	GPR3=43	:	GPR 3
:439	:	GPR4=44	:	GPR 4
:440	:	GPR5=45	:	GPR 5
:441	:	GPR6=46	:	GPR 6
:442	:	CM.SP=46	:	IN COMPATIBILITY MODE THIS IS THE STACK POINTER.
:443	:	GPR7=47	:	GPR 7
:444	:	CM.PC=47	:	IN COMPATIBILITY MODE THIS IS THE PC.
:445	:	GPR8=48	:	GPR 8
:446	:	GPR9=49	:	GPR 9
:447	:	GPR10=4A	:	GPR 10
:448	:	GPR11=4B	:	GPR 11
:449	:	GPR12=4C	:	GPR 12
:450	:	AP=4C	:	ARGUMENT POINTER
:451	:	GPR13=4D	:	GPR 13
:452	:	FP=4D	:	FRAME POINTER
:453	:	SP=4E	:	GPR 14
:454	:	TO=4F	:	TEMP LOCATION 0
:455	:			
:456	:	ZERO=50	:	ZERO CONSTANT
:457	:	#3=51	:	CONSTANT
:458	:	#5=52	:	CONSTANT
:459	:	#6=53	:	CONSTANT
:460	:	#7=54	:	CONSTANT
:461	:	#9=55	:	CONSTANT
:462	:	#B=56	:	CONSTANT
:463	:	#C=57	:	CONSTANT
:464	:	#D=58	:	CONSTANT
:465	:	#E=59	:	CONSTANT
:466	:	#F=5A	:	CONSTANT
:467	:	#13=5B	:	CONSTANT
:468	:	#1F=5C	:	CONSTANT
:469	:	#34=5D	:	CONSTANT
:470	:	#3B=5E	:	CONSTANT
:471	:	#3F=5F	:	CONSTANT
:472	:	#4B=60	:	CONSTANT
:473	:	#66=61	:	CONSTANT
:474	:	#A0=62	:	CONSTANT
:475	:	#F0=63	:	CONSTANT
:476	:	#FF=64	:	CONSTANT
:477	:	#1FF=65	:	CONSTANT
:478	:	#FFF=66	:	CONSTANT
:479	:	#2001=67	:	CONSTANT
:480	:	#3000=68	:	CONSTANT
:481	:	#7F80=69	:	CONSTANT
:482	:	#C000=6A	:	CONSTANT
:483	:	#FFE0=6B	:	CONSTANT


```

:484 : #FFFF=6C : CONSTANT
:485 : #1F0000=6D : CONSTANT
:486 : #200001=6E : CONSTANT
:487 : #F27000=6F : CONSTANT
:488 : #3000000=70 : CONSTANT
:489 : #41F0000=71 : CONSTANT
:490 : #7000000=72 : CONSTANT
:491 : #3FFFFFFE00=73 : CONSTANT
:492 : #7F800000=74 : CONSTANT
:493 : #7FFFFFFE00=75 : CONSTANT
:494 : #7FFFFFF0E=76 : CONSTANT
:495 : #BF800001=77 : CONSTANT
:496 :
:497 : GPR.OS=78 : HARDWARE INDEX TO GPRS
:498 : BIT.OS(3-0)=79 : 16 LOCATION HARDWARE INDEX TO BIT MASKS
:499 : BIT.OS(4-0)=7B : 32 LOCATION HARDWARE INDEX TO BIT MASKS
:500 : OS=7C : OS REGISTER
:501 : DEC.CON=7D : DECIMAL CARRIES
:502 : SIZE=7E : SIZE REGISTER
:503 : MDT= 7E : MEMORY REFERENCE DATA SIZE
:504 : INTERRUPT.VEC=7E : HARDWARE INTERRUPT VECTOR
:505 :
:506 : THIS LOCATION IS THE HARDWARE CONDITION CODES. THE CC'S CAN BE
:507 : READ OR ARE WRITTEN HERE. NO OTHER BITS IN THE PSL ARE EFFECTED.
:508 :
:509 : PSL.CC=7F : PSL CONDITION CODES
:510 :
:511 : THESE ADDRESSES ARE ONLY ACCESSIBLE WITH THE MOV MICRO INSTRUCTION.
:512 :
:513 : KSP=80 : KERNEL STACK POINTER
:514 : ESP=81 : EXECUTIVE STACK POINTER
:515 : SSP=82 : SUPERVISOR STACK POINTER
:516 : USP=83 : USER STACK POINTER
:517 : ISP=84 : INTERRUPT STACK POINTER
:518 : ASTLVL=85 : AST LEVEL
:519 : PCBB=86 : PROCESS CONTROL BLOCK BASE
:520 : SCBB=87 : SYSTEM CONTROL BLOCK BASE
:521 : SISR=88 : SOFTWARE INTERRUPT SUMMARY REGISTER
:522 :
:523 : CONSOLE.COMMAND=89 : ONE BYTE COMMAND.
:524 : CONSOLE.MODIFIER=8A : ONE BYTE MODIFIER.
:525 : CONSOLE.ADDRESS=8B : FOUR BYTES OF ADDRESS.
:526 : CONSOLE.DATA=8C : FOUR BYTES OF DATA.
:527 : CONSOLE.STATUS=8D : ONE BYTE OF STATUS.
:528 : PSEUDO.NICR=8E : COPY OF NICR THAT 8085 HAS.
:529 : PSEUDO.ICR=8F : COPY OF ICR THAT 8085 HAS.
:530 : DEBUG.SAVE.WRO=90 : TEMPORARY SAVE WRO.
:531 : DEBUG.SAVE.WR1=91 : TEMPORARY SAVE WR1.
:532 : DEBUG.SAVE.ALU.CC=92 : TEMPORARY SAVE CONDITION CODES.
:533 : MEMORY.SIZE=93 : MEMORY SIZE OF THE MACHINE.
:534 : CRD.LOG=94 : INFORMATION ABOUT LAST SOFT MEMORY ERROR.
:535 : PACKET.A=95 : TOP OF PACKET.
:536 : PACKET.B=96 : BOTTOM OF PACKET.
:537 : DEBUG.SAVE.ADDRESS=97 : ADDRESS FOR EXAMINE/DEPOSIT.
:538 :
    
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

FIELD DEFINITIONS F 3-G 2
MICRO2 1L(03) 3-MAR-82 09:34:33
FIELD DEFINITIONS FOR MAIN MICRO WORD

Fiche 1 Frame G2
ENKCA Micro-diagnostic for DAP module
VER 00.08

Sequence 19
Rev 01.00
Page 13

:539	:	SAVED.MDT=99	:	SAVED ADDRESS OF MDT FOR MMIE OPTIMIZATION.
:540	:	POBR=9A	:	P0 BASE REGISTER
:541	:	POLR=9B	:	P0 LENGTH REGISTER
:542	:	P1BR=9C	:	P1 BASE REGISTER
:543	:	P1LR=9D	:	P1 LENGTH REGISTER
:544	:	SBR=9E	:	SYSTEM BASE REGISTER
:545	:	SLR=9F	:	SYSTEM LENGTH REGISTER
:546	:		:	
:547	:	PORT0=0A0	:	RESERVED FOR PORT.
:548	:	PORT1=0A1	:	RESERVED FOR PORT.
:549	:	PORT2=0A2	:	RESERVED FOR PORT.
:550	:	PORT3=0A3	:	RESERVED FOR PORT.
:551	:	PORT4=0A4	:	RESERVED FOR PORT.
:552	:	PORT5=0A5	:	RESERVED FOR PORT.
:553	:	PORT6=0A6	:	RESERVED FOR PORT.
:554	:	PORT7=0A7	:	RESERVED FOR PORT.
:555	:	PORT8=0A8	:	RESERVED FOR PORT.
:556	:	PORT9=0A9	:	RESERVED FOR PORT.
:557	:	PORT10=0AA	:	RESERVED FOR PORT.
:558	:	PORT11=0AB	:	RESERVED FOR PORT.
:559	:	PORT12=0AC	:	RESERVED FOR PORT.
:560	:	PORT13=0AD	:	RESERVED FOR PORT.
:561	:	PORT14=0AE	:	RESERVED FOR PORT.
:562	:	PORT15=0AF	:	RESERVED FOR PORT.
:563	:	PORT16=0B0	:	RESERVED FOR PORT.
:564	:	PORT17=0B1	:	RESERVED FOR PORT.
:565	:	PORT18=0B2	:	RESERVED FOR PORT.
:566	:	PORT19=0B3	:	RESERVED FOR PORT.
:567	:	PORT20=0B4	:	RESERVED FOR PORT.
:568	:	PORT21=0B5	:	RESERVED FOR PORT.
:569	:	PORT22=0B6	:	RESERVED FOR PORT.
:570	:	PORT23=0B7	:	RESERVED FOR PORT.
:571	:	PORT24=0B8	:	RESERVED FOR PORT.
:572	:	PORT25=0B9	:	RESERVED FOR PORT.
:573	:	PORT26=0BA	:	RESERVED FOR PORT.
:574	:	PORT27=0BB	:	RESERVED FOR PORT.
:575	:	PORT28=0BC	:	RESERVED FOR PORT.
:576	:	PORT29=0BD	:	RESERVED FOR PORT.
:577	:	PORT30=0BE	:	RESERVED FOR PORT.
:578	:	PORT31=0BF	:	RESERVED FOR PORT.
:579	:		:	
:580	:	XGPR0=0C0	:	BACKUP COPY OF GPR 0
:581	:	XGPR1=0C1	:	BACKUP COPY OF GPR 1
:582	:	XGPR2=0C2	:	BACKUP COPY OF GPR 2
:583	:	XGPR3=0C3	:	BACKUP COPY OF GPR 3
:584	:	XGPR4=0C4	:	BACKUP COPY OF GPR 4
:585	:	XGPR5=0C5	:	BACKUP COPY OF GPR 5
:586	:	XGPR6=0C6	:	BACKUP COPY OF GPR 6
:587	:	XGPR7=0C7	:	BACKUP COPY OF GPR 7
:588	:	XGPR8=0C8	:	BACKUP COPY OF GPR 8
:589	:	XGPR9=0C9	:	BACKUP COPY OF GPR 9
:590	:	XGPR10=0CA	:	BACKUP COPY OF GPR 10
:591	:	XGPR11=0CB	:	BACKUP COPY OF GPR 11
:592	:	XGPR12=0CC	:	BACKUP COPY OF GPR 12
:593	:	XGPR13=0CD	:	BACKUP COPY OF GPR 13


```

:594 : XSP=0CE ; BACKUP COPY OF GPR 14
:595 :
:596 : #14=0D0 ; CONSTANT
:597 : #18=0D1 ; CONSTANT
:598 : #1C=0D2 ; CONSTANT
:599 : #24=0D3 ; CONSTANT
:600 : #28=0D4 ; CONSTANT
:601 : #2C=0D5 ; CONSTANT
:602 : #2D=0D6 ; CONSTANT
:603 : #30=0D7 ; CONSTANT
:604 : #F8=0D8 ; CONSTANT
:605 : #FC=0D9 ; CONSTANT
:606 : #2D20=0DA ; CONSTANT
:607 : #140000=0DB ; CONSTANT
:608 : #150000=0DC ; CONSTANT
:609 : #160000=0DD ; CONSTANT
:610 : #170000=0DE ; CONSTANT
:611 : #180000=0DF ; CONSTANT
:612 : #1E0000=0E0 ; CONSTANT
:613 : #40A10=0E1 ; CONSTANT
:614 : #C0000E0=0E2 ; CONSTANT
:615 : #0FFF0000=0E3 ; CONSTANT
:616 : #B020FF00=0E4 ; CONSTANT
:617 : #FFFFFF10=0E5 ; CONSTANT
:618 : DEBUG.SAVE.T1=0E6 ; TEMPORARY SAVE T1.
:619 : DEBUG.SAVE.OS=0E7 ; TEMPORARY SAVE OS.
:620 : DEBUG.SAVE.SIZE=0E8 ; TEMPORARY SAVE SIZE VALUE.
:621 : SAVED.WR0=0E9 ; MM SAVED WR0
:622 : SAVED.WR1=0EA ; MM SAVED WR1
:623 : SAVED.2ND.VAR=0EB ; MM SAVED VAR 2
:624 : SAVED.ALU.CC=0EC ; MM SAVED ALU CONDITION CODES
:625 : SAVED.SIZE=0ED ; MM SAVED LS[SIZE]
:626 : SAVED.OS=0EE ; MM SAVED LS[OS]
:627 : SAVED.REQUEST=0EF ; MM SAVED REQUEST DATA
:628 : SAVED.CRD.LOG=0F0 ; MM SAVED LOCATION
:629 : OS.BAK=0F1 ; USED BY WARM FLOATING POINT.
:630 :
:631 : THIS LOCATION IS FOR USE BY THE MFPR AND MTPR TO THE CONSOLE AND
:632 : TU58 CSR'S. ALL OF THE INTERRUPT ENABLE BITS ARE HEREIN.
:633 :
:634 : CONSOLE/PROGRAM I/O MODE FLAG - BIT<31> 0 - CONSOLE 1 - PROGRAM I/O MODE
:635 :
:636 : CRD RETRY IN PROGRESS - BIT<5> 0 - NOTHING DOING 1 - IN PROGRESS
:637 : MACHINE CHECK IN PROGRESS - BIT<4>
:638 : TU58 TRANSMIT CSR IE - BIT<3>
:639 : TU58 RECEIVER CSR IE - BIT<2>
:640 : CONSOLE TRANSMIT CSR IE - BIT<1>
:641 : CONSOLE TRANSMIT CSR IE - BIT<0>
:642 :
:643 : CONSOLE.CSR.IES=0F2 ; CONSOLE CSR IE'S AND CONSOLE/PROGRAM I/O MODE FLAG.
:644 : IE.TEMP1=0F3 ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION ONE
:645 : IE.TEMP2=0F4 ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION TWO
:646 : IE.TEMP3=0F5 ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION THREE
:647 :
:648 : XGPR.OS=0F8 ; HARDWARE INDEX TO XGPRS
  
```

```

:649 : PORT(3-0)=0F9 : 16 LOCATION HARDWARE INDEX TO PORT REGISTERS.
:650 : PORT(4-0)=0FB : 32 LOCATION HARDWARE INDEX TO PORT REGISTERS.
:651 : CCR=0FC : CONSOLE READ REGISTER
:652 : TIMER=0FD : INTERVAL TIMER VALUE.
:653 : ALU.CC=0FE : ALU CONDITION CODES
:654 :
:655 : THIS LOCATION IS A WRITE ONLY REGISTER. THE FOLLOWING BITS IN THE
:656 : PSL ARE AFFECTED:
:657 :
:658 : COMPATIBILITY MODE - BIT<31>
:659 : CURRENT MODE - BITS<25:24>
:660 : INTERRUPT PRIORITY LEVEL (IPL) - BITS<20:16>
:661 : TRACE TRAP ENABLE (REALLY T OR TP) - BIT<4>
:662 : NEGATIVE CONDITION CODE - BIT<3>
:663 : ZERO CONDITION CODE - BIT<2>
:664 : OVERFLOW CONDITION CODE - BIT<1>
:665 : CARRY CONDITION CODE - BIT<0>
:666 :
:667 : PSL.HW=0FF : HARDWARE PSL BITS
:668 :
:669 :
:670 : DATA PATH CONTROL
:671 :
:672 : THIS FIELD IS BUILT FROM AN ADDRESS IN THE CCODE MEMORY. THE CCODE MEMORY
:673 : CONTAINS DATA PATH MICRO INSTRUCTIONS TO PRODUCE THE DESIRED OPERATION
:674 : FOR THE UCODE MACROS.
:675 :
:676 : DATA PATH CONTROL FIELD FOR THE 'BASIC' MICRO INSTRUCTION
:677 :
:678 : DP/= <21:16> ,.VALIDITY=<DP.VAL> ,.DEFAULT=< .SHIFT[.AND[<SDP/Q.CMP.LS> ,0FF] , -2]>
:679 :
:680 : THIS FIELD IS ONLY USED WITH THE INSTRUCTIONS WHICH CAN HAVE
:681 : XCHG ATTACHED.
:682 :
:683 : DP.SMALL/= <20:16> ,.VALIDITY=<DP.VAL>
:684 :
:685 : EXCHANGE ORIGINAL WR TO LS
:686 :
:687 : XCHG.BIT/= <21> ,.DEFAULT=<0>
:688 :
:689 : YES=1
:690 : NO=0
:691 :
:692 : DATA PATH CONTROL FIELD FOR THE 'EXTENDED' MICRO INSTRUCTION
:693 :
:694 : XDP/= <19:14> ,.VALIDITY=<A.VAL>
:695 :
:696 : DATA PATH CONTROL FOR THE 'MOVE' MICRO INSTRUCTION
:697 :
:698 : MDP/= <19:17> ,.VALIDITY=<XD.VAL>
:699 :
    
```



```

:700 .PAGE
:701 : CONDITION CODE FIELD / DATA PATH CONTEXT
:702
:703 CC/= <6:5>, .VALIDITY=<CC.VAL>, .DEFAULT=<CC/DT(LONG)&HOLD.ALU.CC>
:704
:705 DT(LONG)&HOLD.ALU.CC=0 ; USE LONG CONTEXT(32 BITS) AND HOLD ALU CONDITION CODES
:706 DT(LONG)&SET.ALU.CC=1 ; USE LONG CONTEXT(32 BITS) AND SET ALU CONDITION CODES
:707 DT(SIZE)&SET.ALU.CC=2 ; USE CONTEXT IN THE SIZE REGISTER AND SET ALU CONDITION CODES
:708 DT(SIZE)&MAP.ALU.CC.TO.PSL=3 ; TRANSFER ALU CONDITION CODES TO PSL CONDITION CODES HARDWARE DEPENDENT
:709
:710
:711 : MEMORY DATA TYPE FIELD
:712
:713 MDT/= <6:5>, .VALIDITY=<DT.VAL>
:714
:715 BYTE=0 ; SIZE = BYTE
:716 WORD=1 ; SIZE = WORD
:717 SIZE=2 ; SIZE = CONTENTS OF SIZE REGISTER
:718 LONG=3 ; SIZE = LONG
:719
:720
:721 : SHIFT INPUT FIELD
:722
:723 : THIS CONTROL FIELD DETERMINES THE SHIFT INPUTS FOR THE FOLLOWING OPERATIONS.
:724 : THE DIRECTION OF THE SHIFT IS DETERMINED BY THE 2901 DESTINATION CODE.
:725
:726 SI/= <13:11>, .VALIDITY=<A.VAL>
:727
:728 ROT.WR=0 ; ROTATE WR
:729 ROT.WR.Q=1 ; ROTATE WR AND Q
:730 ASH.WR=2 ; ARITHMETIC SHIFT OF WR
:731 ASH.WR.Q=3 ; ARITHMETIC SHIFT OF WR AND Q
:732 MUL.SHF=4 ; SIGNED MULTIPLY SHIFT OPERATION
:733 UMUL.SHF=5 ; UNSIGNED MULTIPLY SHIFT OPERATION
:734 SHF.WR.Q=6 ; LOGICAL SHIFT WR AND Q
:735
:736
:737 : SKIP MICRO CONTROL FIELD
:738
:739 : THIS FIELD IS VALID FOR ALL INSTRUCTIONS EXCEPT
:740
:741 JUMP MICRO INSTRUCTION
:742 DECODE MICRO INSTRUCTION
:743
:744 SCTL/= <4:0>, .VALIDITY=<.AND[<SCTL.VAL>, <SKIP.PAGE.VAL>]>, .DEFAULT=<SCTL/NO.SKIP>
:745
:746 N.CLR=0 ; ALU N-BIT EQUAL ZERO
:747 NEQ=1 ; ALU Z-BIT EQUAL ZERO
:748 BIT.SET=1 ; ALU Z-BIT EQUAL ZERO
:749 V.CLR=2 ; ALU V-BIT EQUAL ZERO
:750 C.SET=3 ; ALU C-BIT EQUAL ONE
:751 GEQU=3 ; ALU C-BIT EQUAL ONE
:752 NOT.PSL.C=4 ; PSL C EQUAL ZERO
:753 BR.FALSE=5 ; BRANCH INSTRUCTION FALSE
:754 R.DST=6 ; REGISTER MODE FLAG
  
```

:755	NO.INTERRUPT=7	: NO INTERRUPT PENDING
:756	N.SET=8	: ALU N-BIT EQUAL ONE
:757	EQL=9	: ALU Z-BIT EQUAL ONE
:758	BITS.CLR=9	: ALU Z-BIT EQUAL ONE
:759	V.SET=0A	: ALU V-BIT EQUAL ONE
:760	C.CLR=0B	: ALU C-BIT EQUAL ZERO
:761	LSSU=0B	: ALU C-BIT EQUAL ZERO
:762	STATE.ZERO.SET=0C	: STATE REGISTER BIT ZERO TRUE
:763	STATE.ONE.SET=0D	: STATE REGISTER BIT ONE TRUE
:764	STATE.ZERO.CLR=0E	: STATE REGISTER BIT ZERO FALSE
:765	STATE.ONE.CLR=0F	: STATE REGISTER BIT ONE FALSE
:766	RET.NOERR=10	: RETURN+1 IF NO MEMORY ERROR
:767	JMP.Z.CLR=11	: JUMP TO (STACK) IF ALU Z = 0
:768	POP.USTACK=12	: DISCARD TOP STACK ENTRY
:769	JMP.C.SET=13	: JUMP TO (STACK) IF ALU C=1
:770	RETURN=14	: RETURN TO (USTACK)
:771	NO.SKIP=15	: EXECUTE NEXT INSTRUCTION
:772	RETURN+1=16	: RETURN TO (USTACK)+1
:773	LOOP.IF.NO.I.AND.SYNC=17	: JUMP TO (STACK) IF NO INTERRUPT AND NO ACCELERATOR SYNC.
:774	CONSOLE.ATTN=18	: CONSOLE ATTENTION
:775	CONSOLE.ACK=19	: CONSOLE ACKNOWLEDGE
:776	NO.FAST.INTERRUPT=1A	: NO FAST INTERRUPTS PENDING
:777	BACKUP.MASK.VALID=1B	: REGISTER BACKUP MASK VALID.
:778	LSS=1C	: SIGNED LESS THAN.
:779	MEM.REF.OK=1D	: NO MEMORY ERROR TRUE
:780	SKIP=1E	: EXECUTE ADDRESS PLUS ONE
:781	SYNC=1F	: ACCELERATOR SYNCHRONIZATION

: JUMP MICRO CONTROL FIELD

: THIS FIELD IS VALID FOR THE FOLLOWING INSTRUCTIONS

JUMP MICRO INSTRUCTION
 DECODE MICRO INSTRUCTION

JCTL/= <3:0>, .VALIDITY=<.AND[<JCTL.VAL>, <JUMP.PAGE.VAL>]>, .DEFAULT=<JCTL/JUMP.VALID>

:792		
:793	N.CLR=0	: ALU N-BIT EQUAL ZERO
:794	NEQ=1	: ALU Z-BIT EQUAL ZERO
:795	BIT.SET=1	: ALU Z-BIT EQUAL ZERO
:796	V.CLR=2	: ALU V-BIT EQUAL ZERO
:797	C.SET=3	: ALU C-BIT EQUAL ONE
:798	GEQU=3	: ALU C-BIT EQUAL ONE
:799	JUMP=4	: JUMP UNCONDITIONALLY
:800	BR.FALSE=5	: BRANCH INSTRUCTION FALSE
:801	R.DST=6	: REGISTER MODE FLAG
:802	NO.INTERRUPT=7	: NO INTERRUPT PENDING
:803	N.SET=8	: ALU N-BIT EQUAL ONE
:804	EQL=9	: ALU Z-BIT EQUAL ONE
:805	BITS.CLR=9	: ALU Z-BIT EQUAL ONE
:806	V.SET=0A	: ALU V-BIT EQUAL ONE
:807	C.CLR=0B	: ALU C-BIT EQUAL ZERO
:808	LSSU=0B	: ALU C-BIT EQUAL ZERO
:809	JSR=0C	: UNCONDITIONAL JUMP AND PUSH USTACK


```

:810      JSR.VALID=0D      : JUMP AND PUSH USTACK IF IB VALID=1
:811      NO.JUMP.TST=0E   : DO NOT JUMP AND SKIP IF IB VALID=0
:812      JUMP.VALID=0F    : JUMP IF IB VALID=1
:813
:814
:815      : JUMP ADDRESS FIELD
:816      :
:817      JUMP.ADRS/=<17:4>,.ADDRESS..VALIDITY=<JUMP.VAL>
:818
:819
:820      : OS ENABLE
:821      :
:822      OS/=<18>,.VALIDITY=<JUMP.VAL>
:823
:824      NOP=0
:825      WITH.OS=1        : CONCATENATE OS REGISTER TO JUMP ADDRESS
:826
:827
  
```

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

FIELD DEFINITIONS F 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
FIELD DEFINITIONS FOR MAIN MICRO WORD

Fiche 1 Frame M2
ENKCA Micro-diagnostic for DAP module
VER 00.08

Sequence 25
Rev 01.00
Page 19

```
:828 .PAGE
:829 : BACKUP PC
:830
:831 BPC/= <18> , .VALIDITY=<DECODE.VAL> , .DEFAULT=<.CASE[<.EQL[<OPC2/> , <OPC2/DECODE>]]>]OF[0,1]>
:832
:833 NOP=1
:834 SAVE.PC=0 ; WRITE ALU DATA INTO WR[B] (PC+1)
:835
:836
:837 : DECODE ADDRESS FIELD
:838
:839 DEC.ADRS/= <17:12> , .VALIDITY=<DECODE.VAL>
:840
:841
:842 : IR FUNCTION
:843
:844 IFUNC/= <11:9> , .VALIDITY=<DECODE.VAL>
:845
:846 CM.EXEC=0 ; COMPATIBILITY MODE EXECUTION DISPATCH WHEN UPPER BYTE = ZERO
:847 SPEC=0 ; SPECIFIER DISPATCH
:848 CM.IRD=1 ; COMPATIBILITY MODE INSTRUCTION DECODE DISPATCH
:849 FLOAT=1 ; SPECIFIER FLOAT TYPE DISPATCH
:850 VAX.EXEC=2 ; VAX EXECUTION DISPATCH
:851 ASRC=2 ; ADDRESS SPECIFIER DISPATCH
:852 VAX.IRD=3 ; VAX OPCODE DISPATCH
:853 VSRC=4 ; ADDRESS SPECIFIER DISPATCH FOR BIT FIELD INSTRUCTIONS
:854 ESRC=5 ; SPECIFIER DISPATCH IN INDEX MODE
:855 CM.DST=6 ; COMPATIBILITY MODE DESTINATION DISPATCH
:856 CM.SINGLE=7 ; COMPATIBILITY MODE SOP DISPATCH
:857
:858
:859 : INSTRUCTION BUFFER REQUEST
:860
:861 IB.REQ/= <8> , .VALIDITY=<DECODE.VAL> , .DEFAULT=0
:862
:863 NOP=0
:864 IB.REQ=1 ; ENABLE HARDWARE DEPENDENT MEMORY REQUEST
:865
:866 : COMPATIBILITY MODE OPCODE SELECT
:867
:868 CM.HI/= <7> , .VALIDITY=<DECODE.VAL> , .DEFAULT=0
:869
:870 LOW.BYTE=0 ; DECODE IR BITS<7:0>
:871 HI.BYTE=1 ; DECODE IR BITS <15:6>
:872
:873
:874 : LOAD OS REGISTER
:875
:876 LD.OS/= <6> , .VALIDITY=<DECODE.VAL> , .DEFAULT=0
:877
:878 NOP=0
:879 LOAD.OS=1 ; LOAD OS REGISTER FROM I-BUFFER
:880
:881
:882 : REGISTER DESTINATION FLAG ENABLE
```



```

:883 ;
:884 R.DST/=<5>,.VALIDITY=<DECODE.VAL>,.DEFAULT=0
:885
:886     NOP=0
:887     ENAB=1
:888 ;
:889 ; OPCODE SPECIFIER BIT
:890 ;
:891 OPC.SPEC/=<4>,.VALIDITY=<DECODE.VAL>
:892
:893     CM.EXEC=1          ; COMPATIBILITY MODE EXECUTION DISPATCH
:894     SPEC=0            ; SPECIFIER DISPATCH
:895     CM.IRD=1          ; COMPATIBILITY MODE INSTRUCTION DECODE DISPATCH
:896     FLOAT=0          ; SPECIFIER FLOAT TYPE DISPATCH
:897     VAX.EXEC=1        ; VAX EXECUTION DISPATCH
:898     ASRC=0            ; ADDRESS SPECIFIER DISPATCH
:899     VAX.IRD=1         ; VAX OPCODE DISPATCH
:900     VSRC=0            ; ADDRESS SPECIFIER DISPATCH FOR BIT FIELD INSTRUCTIONS
:901     ESRC=0            ; SPECIFIER DISPATCH IN INDEX MODE
:902     CM.DST=0          ; COMPATIBILITY MODE DESTINATION DISPATCH
:903     CM.SINGLE=0        ; COMPATIBILITY MODE SOP DISPATCH
    
```

```

:904 .PAGE
:905 . MISCELLANEOUS FUNCTIONS
:906
:907
:908 THIS FIELD IS A FLAG FOR STEERING THE MICROWORD AS A MISCELLANEOUS
:909 INSTRUCTION OR AS A PORT INSTRUCTION.
:910
:911 MISC.PORT/=<18>
:912
:913 MISC=0
:914 PORT=1
:915
:916 MISC.0/=<17>,.VALIDITY=<MISC.VAL>
:917
:918 NOP=0 ; NO OPERATION IN ALU OR CONDITION CODES.
:919
:920 MISC.1/=<15:12>,.VALIDITY=<MISC.VAL>
:921
:922 NOP=0F ; NO OPERATION
:923 SET.CP.ATTN=0E ; SET CPU ATTENTION FOR CONSOLE
:924 SET.CP.ACK=0D ; SET CPU ACKNOWLEDGE FOR CONSOLE
:925 CLR.CP.ATTN.AND.ACK=0C ; CLEAR CPU ATTENTION AND CPU ACKNOWLEDGE
:926 SET.HIGH.PAGE=0B ; SET BIT FOR HIGH HALF OF WCS.
:927 CLR.HIGH.PAGE=0A ; CLEAR BIT FOR LOW HALF OF WCS.
:928 SET.PORT.I.O=9 ; SET PORT I/O MODE
:929 SET.ACC.I.O=8 ; SET ACCELERATOR I/O MODE
:930 SET.BACKUP.MASK.VALID=7 ; SET REGISTER BACKUP MASK FLAG
:931 SET.S1=6 ; SET STATE ONE BIT
:932 SET.S0=5 ; SET STATE ZERO BIT
:933 CLR.S1=4 ; CLEAR STATE ONE BIT
:934 CLR.S0=3 ; CLEAR STATE ZERO BIT
:935 SET.S1&CLR.S0=2 ; SET STATE ONE BIT AND CLEAR STATE ZERO BIT
:936 CLR.S1&SET.S0=1 ; CLEAR STATE ONE AND SET STATE ZERO
:937 CLR=0 ; CLEAR STATE ONE AND STATE ZERO BITS.
:938
:939
:940 MISC.2/=<11:9>,.VALIDITY=<MISC.VAL>
:941
:942 NOP=0 ; NO OPERATION
:943 MASK.INTERRUPTS=1 ; MASK ALL INTERRUPTS (FOR DECODE NEXT CYCLE).
:944 ASSERT.DA.AND.PASS.WR=2 ; ASSERT DATA AVAILABLE AND PASS WR[0] TO THE ACCELERATOR OR PORT.
:945 TRAP.ACC=3 ; TRAP ACCELERATOR
:946 READ.ACC.UPC=4 ; READ ACCELERATOR PROGRAM COUNTER
:947 TRANSFER.GRANT=5 ; RECOGNIZE PORT REQUEST.
:948 MASK.HALT.AND.T.BIT=6 ; MASK HALT AND T-BIT TRAPS (FOR DECODE TWO CYCLES LATER)
:949 WRITE.WR.TO.CONSOLE.REGISTER=7 ; SEND BITS <7:0> TO 8085.
:950
:951 MISC.WR/=<8:7>
:952
:953 MISC.WRL/=<8>
:954 MISC.WRR/=<7>
:955
:956 PORT.SELECT/=<17:15>
:957
:958 IDC=7
  
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

; ENKCA.MIC

FIELD DEFINITIONS F 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
FIELD DEFINITIONS FOR MAIN MICRO WORD

C 3

ENKCA Micro-diagnostic for DAP module
VER 00.08

Fiche 1 Frame C3

Sequence 28

Page 22

:959
:960 PORT.FUNC/= <14:13>
:961
:962 READ=0
:963 WRITE=1
:964 CONTROL=2
:965
:966 R.W.FUNC/= <12:10>
:967
:968 CSR=0
:969 DAR=1
:970 DATA.BYTE=2
:971 DATA.WORD=3
:972 PATTERN=4
:973 POSITION=5
:974
:975 CNTL.FUNC/= <12:10>
:976
:977 CLEAR.FIFO.CNTR=0
:978 RESET.BR=1
:979 CLEAR.IDC=3
:980 SET.AUTO=4
:981 CLEAR.AUTO=5
:982 SEL.FIFO.A=6
:983 SEL.FIFO.B=7
:984

```

:985 .PAGE
:986 .MEMORY REQUEST FIELD
:987
:988 THIS FIELD IS USED TO INDICATE THE DESIRED OPERATION TO THE MEMORY CONTROL.
:989 NOTE THAT THIS IS A DUMMY FIELD. IT IS ACTUALLY MADE UP OF TWO FIELDS.
:990 M.FUNC1 AND M.FUNC2
:991
:992 THE FOLLOWING SPECIAL CONSIDERATIONS MUST BE MADE:
:993
:994 ROTATE.BYTE.RIGHT - PERFORMS 9-BIT ROTATE AND THEREFORE THE
:995 ANSWER MUST BE CLEANED UP WITH A ROL WRL].
:996
:997 WORD.SWAP - PERFORMS A 15-BIT ROTATE AND THERE FORE
:998 THE ANSWER MUST BE CLEANED UP WITH A ROL WRL].
:999
:1000 OCTA.WRITE.P - MUST BE LONGWORD ALLIGNED.
:1001
:1002 OCTA.WR.V.WCHK - MUST BE LONGWORD ALLIGNED. THIS DATA CAN
:1003 CROSS PAGE BOUNDARIES
:1004
:1005 MEM.FUNC/=<7>,.VALIDITY=0
:1006
:1007 :
:1008 : PREFETCH=0 : USED TO TAKE VAR AND ADD ONE BEFORE REFERENCE.
:1009 : WRITE.TB=01 : WRITE TRANSLATION BUFFER
:1010 : TEST.V.RCHK=2 : TEST WITH VIRTUAL ADDRESS AND READ ACCESS CHECK
:1011 : READ.P=3 : READ WITH PHYSICAL ADDRESS
:1012 : WRITE.P=4 : WRITE WITH PHYSICAL ADDRESS
:1013 : TEST.V.WCHK=5 : WRITE WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK
:1014 : ROTATE.BYTE.RIGHT=6 : ROTATE ADDRESS DATA 9 BITS RIGHT AND RETURN
:1015 : WORD.SWAP=7 : ROTATE ADDRESS DATA 15 BITS LEFT AND RETURN
:1016 : READ.V.NOCHK=8 : READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK
:1017 : READ.V.WCHK=9 : READ WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK
:1018 : READ.V.RCHK=0A : READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK
:1019 : READ.MAINT.VAR.INC=0B : TEST VAR INCREMENTING.
:1020 : WRITE.V.NOCHK=0C : WRITE WITH VIRTUAL ADDRESS AND NO ACCESS CHECK
:1021 : WRITE.V.WCHK=0D : WRITE WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK
:1022 : IB.FILL=0E : READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK AND FILL IB
:1023 : READ.V.RCHK.IFILL=0E : READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK AND FILL IB
:1024 : WRITE.UBS.MAP=0F : WRITE TO THE UNIBUS MAP.
:1025
:1026 : OCTA.WRITE.P=10 : WRITE OCTA DATA WITH PHYSICAL ADDRESS.
:1027 : WRITE.TB.STEP=11 : WRITE TO TWO TB ENTRIES(WITH CORRECT ADDRESS)
:1028 : READ.UBS.MAP=12 : READ UNIBUS MAP
:1029 : READ.TB=13 : READ TRANSLATION BUFFER
:1030 : READ.MAINT.ADRS=14 : RPUTS VA ON UNIBUS LINES AND READS THIS DATA
:1031 : MAINT.ECC.DATA=15 : TEST ALL ECC FUNCTIONS.
:1032 : READ.CSR=16 : READ CONTROL REGISTER
:1033 : READ.CNTRL.REG=16 : READ CONTROL REGISTER
:1034 : WRITE.CNTRL.REG=17 : WRITE CONTROL REGISTER
:1035 : WRITE.CSR=17 : WRITE CONTROL REGISTER
:1036 : OCTA.READ.P=18 : READ OCTA DATA WITH PHYSICAL ADDRESS.
:1037 : READ.V.WCHK.LOCKED=19 : READ VIRTUAL ADDRESS AND WRITE ACCESS CHECK AND LOCKED
:1038 : OCTA.READ.V.RCHK=1A : READ OCTA WITH VIRTUAL ADDRESS AND READ ACCESS CHECK.
:1039 : READ.MAINT.BR.CHECK=1B : TESTS BRANCHING FUNCTION.
:1040 : ISSUE.BG=1C : ISSUES BUS GRANT (4+ADDRESS<1:0>)
    
```


:1040
:1041
:1042
:1043
:1044
:1045
:1046
:1047
:1048
:1049
:1050

OCTA.WR.V.WCHK=1D
QUAD.READ.V.RCHK=1E
READ.MAINT.UBS=1F

M.FUNC2/=<18:16>,.VALIDITY=<DT.VAL>
M.FUNC1/=<8:7>,.VALIDITY=<DT.VAL>
PAR/=<23>,.DEFAULT=< .PARITY[<OPC2/>,<OS/>,<JUMP.ADRS/>,<JCTL/>]>
.UCODE

: WRITE OCTA WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK.
: READ QUAD WITH VIRTUAL ADDRESS AND READ ACCESS CHECK.
: PUTS VA ON UNIBUS LINES AND READS THIS AS DATA.

```

:1051 .PAGE 'MACRO DEFINITIONS VER 00.02"
:1052
:1053 THE FORMAT FOR THIS GROUP IS TWO ADDRESS WITH A MODIFY DESTINATION.
:1054 THE FIRST OPERAND IS COMBINED WITH THE SECOND OPERAND AND THE
:1055 RESULT WRITTEN TO THE SECOND OPERAND. CMP AND BIT INSTRUCTIONS ARE
:1056 READ ONLY.
:1057
:1058 DURING ALL ARITHMETIC AND LOGICAL INSTRUCTIONS THE ALU CC'S
:1059 CAN BE LOADED BY ADDING THE
:1060
:1061 DT(SIZE)&SET.ALU.CC
:1062
:1063 MACRO TO THE MICROWORD. THIS SPECIFIES THAT ALU CC'S ARE LOADED
:1064 WITH THE 2901 CONDITIONS. NO EXTERNAL MUNGING IS DONE. IF THE
:1065 DATA OPERATION IS WITH BYTES THEN THE CC'S ARE SET
:1066 ACCORDING TO BITS 7-0. WORD USES BITS 15-0 AND LONGWORD USES BITS
:1067 31-0. THE FOLLOWING IS A DESCRIPTION OF THE ALU CC'S VALUE
:1068 FOR EACH FUNCTION:
:1069
:1070 ADD -- BINARY ADDITION OF THE TWO OPERANDS
:1071
:1072 N - SIGN BIT
:1073 Z - SET IF ANSWER IS ZERO.
:1074 V - ARITHMETIC OVERFLOW
:1075 C - CARRY
:1076
:1077 SUB -- BINARY ADDITION OF FIRST OPERAND AND TWO'S COMPLEMENT OF THE SECOND OPERAND.
:1078
:1079 N - SIGN BIT
:1080 Z - SET IF ANSWER IS ZERO.
:1081 V - ARITHMETIC OVERFLOW
:1082 C - COMPLEMENT OF THE BORROW.
:1083
:1084 BIS -- LOGICAL OR OF THE TWO OPERANDS.
:1085
:1086 N - SIGN BIT
:1087 Z - SET IF ANSWER IS ZERO
:1088 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1089 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1090
:1091 BIC -- ONE'S COMPLEMENT OF FIRST OPERAND ANDED WITH SECOND OPERAND.
:1092
:1093 N - SIGN BIT
:1094 Z - SET IF ANSWER IS ZERO
:1095 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1096 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1097
:1098 AND -- LOGICAL AND OF THE TWO OPERANDS.
:1099
:1100 N - SIGN BIT
:1101 Z - SET IF ANSWER IS ZERO
:1102 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1103 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1104
:1105 XOR -- LOGICAL EXCLUSIVE OR OF THE TWO OPERANDS.

```

```

:1106 :
:1107 :
:1108 :
:1109 :
:1110 :
:1111 :
:1112 :
:1113 :
:1114 :
:1115 :
:1116 :
:1117 :
:1118 :
:1119 :
:1120 :
:1121 :
:1122 :
:1123 :
:1124 :
:1125 :
:1126 :
:1127 :
:1128 :
:1129 :
:1130 :
:1131 :

```

N - SIGN BIT
 Z - SET IF ANSWER IS ZERO
 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)

CMP -- PERFORM BINARY ADDITION OF FIRST OPERAND AND TWO'S COMPLEMENT OF SECOND OPERAND BUT DO NOT STORE.

N - SIGN BIT
 Z - SET IF ANSWER IS ZERO.
 V - ARITHMETIC OVERFLOW
 C - CARRY

BIT -- PERFORM LOGICAL AND OF THE TWO OPERANDS BUT DO NOT STORE.

N - SIGN BIT
 Z - SET IF ANSWER IS ZERO
 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)

SWAP -- SWITCH THE TWO VALUES.

N - SIGN BIT OF VALUE TO WR.
 Z - SET IF ANSWER IS ZERO OF VALUE TO WR.
 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)


```

:1132 .PAGE
:1133
:1134     MACROS FOR THE FORM OF WR[B]<-- WR[B] OPERATE LS[D]
:1135
:1136 ADD  LS[] TO  WR[]      'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/.SHIFT[<.AND[<SDP/LS.PLUS.WR>,07F]>,-2]>'
:1137 SUB  LS[] FROM WR[]    'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/.SHIFT[<.AND[<SDP/LS.FROM.WR>,07F]>,-2]>'
:1138 BIS  LS[] TO  WR[]    'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/.SHIFT[<.AND[<SDP/LS.OR.WR>,OFF]>,-2]>'
:1139 BIC  LS[] TO  WR[]    'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/.SHIFT[<.AND[<SDP/LS.MASK.WR>,OFF]>,-2]>'
:1140 AND  LS[] TO  WR[]    'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/.SHIFT[<.AND[<SDP/LS.AND.WR>,07F]>,-2]>'
:1141 XOR  LS[] TO  WR[]    'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/.SHIFT[<.AND[<SDP/LS.XOR.WR>,07F]>,-2]>'
:1142
:1143 CMP  LS[] WITH WR[]   'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/.SHIFT[<.AND[<SDP/LS.CMP.WR>,OFF]>,-2]>'
:1144 BIT  LS[] WITH WR[]   'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/.SHIFT[<.AND[<SDP/WR.BIT.LS>,OFF]>,-2]>'
:1145 SWAP LS[] WITH WR[]   'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/.SHIFT[<.AND[<SDP/SWAP.LS.WR>,OFF]>,-2]>'
:1146
:1147     THE FOLLOWING MACROS IS TO BE USED WITH ADD,SUB,
:1148     AND,XOR LS[] TO WR[]
:1149
:1150     IT WILL TAKE THE ORIGINAL CONTENTS OF WR[]
:1151     AND PUT IT IN LS[].
:1152
:1153 XCHG      'XCHG.BIT/YES'
:1154
:1155     MACROS FOR THE FORM OF LS[D]<-- LS[D] OPERATE Q-REGISTER
:1156
:1157 ADD  Q TO  LS[]      'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/Q.PLUS.LS>,OFF]>,-2]>'
:1158 SUB  Q FROM LS[]    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/Q.FROM.LS>,OFF]>,-2]>'
:1159 BIS  Q TO  LS[]    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/Q.OR.LS>,OFF]>,-2]>'
:1160 AND  Q TO  LS[]    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/Q.AND.LS>,OFF]>,-2]>'
:1161 XOR  Q TO  LS[]    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/Q.XOR.LS>,OFF]>,-2]>'
:1162
:1163 CMP  Q WITH LS[]   'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/Q.CMP.LS>,OFF]>,-2]>'
:1164 BIT  Q WITH LS[]   'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/LS.BIT.Q>,OFF]>,-2]>'
:1165
:1166     MACROS FOR THE FORM OF Q-REGISTER<-- Q-REGISTER OPERATE LS[D]
:1167
:1168 ADD  LS[] TO  Q      'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/LS.PLUS.Q>,OFF]>,-2]>'
:1169 SUB  LS[] FROM Q    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/LS.FROM.Q>,OFF]>,-2]>'
:1170 BIS  LS[] TO  Q    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/LS.OR.Q>,OFF]>,-2]>'
:1171 BIC  LS[] TO  Q    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/LS.MASK.Q>,OFF]>,-2]>'
:1172 AND  LS[] TO  Q    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/LS.AND.Q>,OFF]>,-2]>'
:1173 XOR  LS[] TO  Q    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/LS.XOR.Q>,OFF]>,-2]>'
:1174
:1175 CMP  LS[] WITH Q    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/LS.CMP.Q>,OFF]>,-2]>'
:1176 BIT  LS[] WITH Q    'OPC/BASIC,D.ADRS/@1,DP/.SHIFT[<.AND[<SDP/LS.BIT.Q>,OFF]>,-2]>'
:1177
:1178     MACROS FOR THE FORM OF LS[D]<-- LS[D] OPERATE WR[B]
:1179
:1180 ADD  WR[] TO  LS[]    'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/.SHIFT[<.AND[<SDP/WR.PLUS.LS>,OFF]>,-2]>'
:1181 SUB  WR[] FROM LS[]  'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/.SHIFT[<.AND[<SDP/WR.FROM.LS>,OFF]>,-2]>'
:1182 BIS  WR[] TO  LS[]    'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/.SHIFT[<.AND[<SDP/WR.OR.LS>,OFF]>,-2]>'
:1183 AND  WR[] TO  LS[]    'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/.SHIFT[<.AND[<SDP/WR.AND.LS>,OFF]>,-2]>'
:1184 XOR  WR[] TO  LS[]    'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/.SHIFT[<.AND[<SDP/WR.XOR.LS>,OFF]>,-2]>'
:1185
:1186 CMP  WR[] WITH LS[]  'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/.SHIFT[<.AND[<SDP/WR.CMP.LS>,OFF]>,-2]>'
    
```

```
:1187 BIT WR[] WITH LS[]      'OPC1/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.BIT.LS>,OFF]>,-2]>'
:1188 SWAP WR[] WITH LS[]    'SWAP LS[@2] WITH WR[@1]'
:1189 :
:1190 :      MACROS FOR THE FORM OF WR[B]<-- WR[B] OPERATE WR[A]
:1191 :
:1192 ADD WR[] TO WR[]        'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.PLUS.WR>,03F]>'
:1193 SUB WR[] FROM WR[]      'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.FROM.WR>,03F]>'
:1194 BIS WR[] TO WR[]        'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.OR.WR>,03F]>'
:1195 BIC WR[] TO WR[]        'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.MASK.WR>,03F]>'
:1196 AND WR[] TO WR[]        'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.AND.WR>,03F]>'
:1197 XOR WR[] TO WR[]        'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.XOR.WR>,03F]>'
:1198 :
:1199 CMP WR[] WITH WR[]      'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.CMP.WR>,03F]>'
:1200 BIT WR[] WITH WR[]      'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.BIT.WR>,03F]>'
:1201 :
:1202 :      MACROS FOR THE FORM OF WR[B]<-- WR[B] OPERATE Q-REGISTER
:1203 :
:1204 ADD Q TO WR[]           'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.PLUS.WR>,03F]>'
:1205 SUB Q FROM WR[]        'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.FROM.WR>,03F]>'
:1206 BIS Q TO WR[]          'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.OR.WR>,03F]>'
:1207 AND Q TO WR[]          'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.AND.WR>,03F]>'
:1208 XOR Q TO WR[]          'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.XOR.WR>,03F]>'
:1209 :
:1210 CMP Q WITH WR[]         'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.CMP.WR>,03F]>'
:1211 BIT Q WITH WR[]        'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.BIT.WR>,03F]>'
:1212 :
:1213 :      MACROS FOR THE FORM OF Q-REGISTER<-- Q-REGISTER OPERATE WR[B]
:1214 :
:1215 ADD WR[] TO Q           'OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/WR.PLUS.Q>,03F]>'
:1216 SUB WR[] FROM Q        'OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/WR.FROM.Q>,03F]>'
:1217 BIS WR[] TO Q          'OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/WR.OR.Q>,03F]>'
:1218 BIC WR[] TO Q          'OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/WR.MASK.Q>,03F]>'
:1219 AND WR[] TO Q          'OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/WR.AND.Q>,03F]>'
:1220 XOR WR[] TO Q          'OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/WR.XOR.Q>,03F]>'
:1221 :
:1222 CMP WR[] WITH Q         'OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/WR.CMP.Q>,03F]>'
:1223 BIT WR[] WITH Q        'OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/Q.BIT.WR>,03F]>'
```



```

:1224 .PAGE
:1225
:1226 THE FORMAT OF THIS GROUP IS THREE ADDRESS TYPE. THE FIRST AND SECOND
:1227 OPERANDS ARE READ AND THE RESULT IS WRITTEN INTO THE THIRD OPERAND.
:1228
:1229 MACROS FOR THE FORM OF Q-REGISTER<-- WR[B] OPERATE WR[A]
:1230
:1231 ADD WR[] PLUS WR[] TO Q      'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.PLUS.WR.Q>,03F]>'
:1232 SUB WR[] FROM WR[] TO Q    'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.FROM.WR.Q>,03F]>'
:1233 BIS WR[] WITH WR[] TO Q    'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.OR.WR.Q>,03F]>'
:1234 BIC WR[] WITH WR[] TO Q    'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.MASK.WR.Q>,03F]>'
:1235 AND WR[] WITH WR[] TO Q    'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.AND.WR.Q>,03F]>'
:1236 XOR WR[] WITH WR[] TO Q    'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.XOR.WR.Q>,03F]>'
:1237
:1238 MACROS FOR THE FORM OF Q-REGISTER<-- WR[B] OPERATE LS[D]
:1239
:1240 ADD WR[] PLUS LS[] TO Q      'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.LS.Q>,OFF]>,-2]>'
:1241 SUB WR[] FROM LS[] TO Q    'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.FROM.LS.Q>,OFF]>,-2]>'
:1242 BIS WR[] WITH LS[] TO Q    'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.OR.LS.Q>,OFF]>,-2]>'
:1243 AND WR[] WITH LS[] TO Q    'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.AND.LS.Q>,OFF]>,-2]>'
:1244 XOR WR[] WITH LS[] TO Q    'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.XOR.LS.Q>,OFF]>,-2]>'
:1245
:1246 MACROS FOR THE FORM OF Q-REGISTER <-- LS[D] OPERATE WR[B]
:1247
:1248 ADD LS[] PLUS WR[] TO Q      'ADD WR[@2] PLUS LS[@1] TO Q'
:1249 SUB LS[] FROM WR[] TO Q    'OPC/BASIC,B.ADRS/@2,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.FROM.WR.Q>,OFF]>,-2]>'
:1250 BIS LS[] WITH WR[] TO Q    'BIS WR[@2] WITH LS[@1] TO Q'
:1251 AND LS[] WITH WR[] TO Q    'AND WR[@2] WITH LS[@1] TO Q'
:1252 XOR LS[] WITH WR[] TO Q    'XOR WR[@2] WITH LS[@1] TO Q'
:1253
:1254
    
```



```

:1255 .PAGE
:1256 : SPECIAL FUNCTION ADD/SUB OPERATIONS
:1257
:1258 ADD (WR[] TO WR[])+1      'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.PLUS.WR+1>,03F]>'
:1259 ADD (LS[] TO WR[])+1      'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/LS.PLUS.WR+1>,OFF]>,-2]>'
:1260 ADD (WR[] TO LS[])+1      'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.LS+1>,OFF]>,-2]>'
:1261
:1262 ADD OS PLUS WR[] TO LS[]  'OPC1/MOVE,B.ADRS/@1,XD.ADRS/@2,MDP/<.SHIFT[<.AND[<SDP/WR.PLUS.OS>,3F]>,-3]>'
:1263
:1264 SUB (WR[] FROM WR[])-1     'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.FROM.WR-1>,03F]>'
:1265 SUB (LS[] FROM WR[])-1     'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/LS.FROM.WR-1>,OFF]>,-2]>'
:1266 SUB (WR[] FROM LS[])-1     'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.FROM.LS-1>,OFF]>,-2]>'
:1267
:1268
:1269 SUB WRB[] FROM WRA[] TO WRB[] 'OPC1/EXTENDED,B.ADRS/@1,A.ADRS/@2,XDP/<.AND[<SDP/WR.FROM.WR.WR>,03F]>'
:1270 SUB LS[] FROM WR[] TO LS     'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/LS.FROM.WR.LS>,OFF]>,-2]>'
:1271 SUB WR[] FROM LS[] TO WR     'OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WRA.FROM.LS.WRB>,OFF]>,-2]>'
:1272
:1273 SUB WRA[] FROM Q TO WRB[]    'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.FROM.Q.WR>,03F]>'
:1274 SUB LS[] FROM Q TO LS[]     'OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.FROM.Q.LS>,OFF]>,-2]>'
:1275 SUB Q FROM WR[] TO Q         'OPC1/EXTENDED,B.ADRS/@1,XDP/<.AND[<SDP/Q.FROM.WR.Q>,03F]>'
:1276 SUB Q FROM LS[] TO Q        'OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.FROM.LS.Q>,OFF]>,-2]>'
:1277
:1278 ADD Q PLUS WR[] TO LS[]      'OPC/BASIC,D.ADRS/@2,B.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.PLUS.WR.TO.LS>,OFF]>,-2]>'
:1279 SUB Q FROM WR[] TO LS[]     'OPC/BASIC,D.ADRS/@2,B.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.FROM.WR.TO.LS>,OFF]>,-2]>'
:1280 ADD Q PLUS LS[] TO WR[]     'OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/Q.PLUS.LS.TO.WR>,OFF]>,-2]>'
:1281
:1282 ADD Q TO WR[] XCHG TO LS[]  'OPC/BASIC,D.ADRS/@2,B.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.Q.XCHG>,OFF]>,-2]>'
:1283

```

```

:1284 .PAGE
:1285 .MOVE MACRO INSTRUCTIONS
:1286
:1287     DURING THE MOVE INSTRUCTIONS THE ALU CC'S CAN BE LOADED BY
:1288     ADDING THE
:1289
:1290         DT(SIZE)&SET.ALU.CC
:1291
:1292     MACRO TO THE MICROWORD. THE FOLLOWING IS A DESCRIPTION OF THE ALU
:1293     CC'S VALUE FOR EACH FUNCTION:
:1294
:1295     MOV -- PUT FIRST OPERAND ON TOP OF THE SECOND OPERAND.
:1296
:1297     N - SIGN BIT
:1298     Z - SET IF ANSWER IS ZERO
:1299     V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1300     C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1301
:1302     MCOM -- PUT ONE'S COMPLEMENT OF THE FIRST OPERAND ON TOP OF THE SECOND OPERAND.
:1303
:1304     N - SIGN BIT
:1305     Z - SET IF ANSWER IS ZERO
:1306     V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1307     C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1308
:1309     MNEG -- PUT TWO'S COMPLEMENT OF THE FIRST OPERAND ON TOP OF THE SECOND OPERAND.
:1310
:1311     N - SIGN BIT
:1312     Z - SET IF ANSWER IS ZERO.
:1313     V - ARITHMETIC OVERFLOW
:1314     C - CARRY
:1315     MOV Q TO LSC[]      "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/MOV.Q.LS>,OFF]>,-2]>"
:1316     MOV Q TO WR[]      "OPC1/EXTENDED,B.ADRS/@1,XDP/<.AND[<SDP/MOV.Q.WR>,03F]>"
:1317     MOV LSC[] TO Q      "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/MOV.LS.Q>,OFF]>,-2]>"
:1318     MOV Q TO WR[] XCHG TO LSC[] "OPC/BASIC,D.ADRS/@2,B.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/MOV.Q.WR&WR.LS>,OFF]>,-2]>"
:1319
:1320     MOV IB.DATA TO OS    "OPC2/DECODE,IFUNC/SPEC,R.DST/NOP,IB.REQ/IB.REQ,JCTL/NO.JUMP,TST,LD.OS/LOAD.OS"
:1321     MOV CM.IB.DATA TO OS "OPC2/DECODE,IFUNC/SPEC,R.DST/NOP,IB.REQ/NOP,JCTL/NO.JUMP,TST,LD.OS/LOAD.OS"
:1322
:1323     MOV MEM.DATA TO WR[] "OPC1/MOVE,B.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.MEM.WR>,3F]>,-3]>,XD.ADRS/WR.BACKUP"
:1324     MOV MEM.DATA TO WR[] XCHG TO LSC[] "OPC1/MOVE,B.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.MEM.WR>,3F]>,-3]>,XD.ADRS/@2"
:1325     MOV MEM.DATA TO LSC[] "OPC1/MOVE,XD.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.MEM.LS>,3F]>,-3]>"
:1326     MOV LSC[] TO WR[]    "OPC1/MOVE,XD.ADRS/@1,B.ADRS/@2,MDP/<.SHIFT[<.AND[<SDP/MOV.LS.WR>,3F]>,-3]>"
:1327     MOV WR[] TO LSC[]    "OPC1/MOVE,B.ADRS/@1,XD.ADRS/@2,MDP/<.SHIFT[<.AND[<SDP/MOV.WR.LS>,3F]>,-3]>"
:1328     MOV WR[] TO WR[]    "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.PLUS.0.TO.WR>,03F]>"
:1329     MOV WR[] TO Q       "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.OR.WR.Q>,03F]>"
:1330     MOV XWR[] TO Q      "OPC1/MOVE,XB.ADRS/@1,XD.ADRS/0,MDP/<.SHIFT[<.AND[<SDP/MOV.XWR.Q>,3F]>,-3]>"
:1331
:1332     MOV FPA TO LSC[]     "OPC1/MOVE,XD.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.ACC.LS>,3F]>,-3]>"
:1333     MOV PORT TO LSC[]   "OPC1/MOVE,XD.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.ACC.LS>,3F]>,-3]>,CC/DT(LONG)&HOLD.ALU.CC"
:1334
:1335     MCOM WR[] TO LSC[]   "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.COM.LS>,OFF]>,-2]>"
:1336     MCOM LSC[] TO WR[]  "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/LS.COM.WR>,OFF]>,-2]>"
:1337     MNEG WR[] TO LSC[]  "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.NEG.LS>,OFF]>,-2]>"
:1338     MNEG LSC[] TO WR[]  "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/LS.NEG.WR>,OFF]>,-2]>"
    
```

```

:1339 MNEG WR[] TO WR[]      'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.NEG.WR>,03F]>'
:1340 MCOM WR[] TO WR[]    'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.COM.WR>,03F]>'
:1341 MINC WR[] TO WR[]    'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.INC.WR>,03F]>'
:1342 MDEC WR[] TO WR[]    'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.DEC.WR>,03F]>'
:1343 MNEG Q TO LSC[]      'OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.NEG.LS>,OFF]>,-2]>'
  
```



```

:1344 PAGE
:1345
:1346 SINGLE OPERAND OPERATIONS
:1347
:1348 THIS FORM OF INSTRUCTION PERFORMS THE DESIRED OPERATION ON THE OPERAND
:1349 SPECIFIED.
:1350
:1351     CLEAR
:1352     NEGATE
:1353     INCREMENT
:1354     DECREMENT
:1355     COMPLEMENT
:1356     TEST
:1357
:1358 DURING THE SINGLE OPERAND INSTRUCTIONS THE ALU CC'S CAN BE
:1359 LOADING USING THE
:1360
:1361     DT(SIZE)&SET.ALU.CC
:1362
:1363 MACRO IN THE MICROWORD. THE FOLLOWING IS A DESCRIPTION OF THE
:1364 ALU CC'S FOR EACH FUNCTION:
:1365
:1366 CLR -- STORE A ZERO IN THE OPERAND.
:1367
:1368 N - SIGN BIT
:1369 Z - SET IF ANSWER IS ZERO
:1370 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1371 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1372
:1373 NEG -- PERFORM TWO'S COMPLEMENT OF THE OPERAND.
:1374
:1375 N - SIGN BIT
:1376 Z - SET IF ANSWER IS ZERO.
:1377 V - ARITHMETIC OVERFLOW
:1378 C - CARRY
:1379
:1380 INC -- ADD ONE TO THE OPERAND.
:1381
:1382 N - SIGN BIT
:1383 Z - SET IF ANSWER IS ZERO.
:1384 V - ARITHMETIC OVERFLOW
:1385 C - CARRY
:1386
:1387 DEC -- SUBTRACT ONE FROM THE OPERAND.
:1388
:1389 N - SIGN BIT
:1390 Z - SET IF ANSWER IS ZERO.
:1391 V - ARITHMETIC OVERFLOW
:1392 C - CARRY
:1393
:1394 COM -- PERFORM ONE'S COMPLEMENT OF THE OPERAND (XNOR WITH ZERO).
:1395
:1396 N - SIGN BIT
:1397 Z - SET IF ANSWER IS ZERO.
:1398 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
    
```

```

:1399 : C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
:1400 :
:1401 : TST -- ADD ZERO TO THE OPERAND TO SET THE CONDITION CODES.
:1402 :
:1403 : N - SIGN BIT
:1404 : Z - SET IF ANSWER IS ZERO
:1405 : V - ARITHMETIC OVERFLOW (IN THIS CASE ALWAYS ZERO)
:1406 : C - CARRY (IN THIS CASE ZERO)
:1407 :
:1408 :
:1409 CLR Q "OPC1/EXTENDED,A.ADRS/0,B.ADRS/0,XDP/<.AND[<SDP/WR.XOR.WR.Q>,03F]>"
:1410 CLR LS[] "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/CLR.LS>,OFF]>,-2]>"
:1411 CLR WR[] "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.XOR.WR>,03F]>"
:1412 :
:1413 NEG Q "OPC1/EXTENDED,XDP/<.AND[<SDP/NEG.Q>,03F]>"
:1414 NEG LS[] "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/NEG.LS>,OFF]>,-2]>"
:1415 NEG WR[] "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.NEG.WR>,03F]>"
:1416 :
:1417 INC Q "OPC1/EXTENDED,XDP/<.AND[<SDP/INC.Q>,03F]>"
:1418 INC LS[] "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/INC.LS>,OFF]>,-2]>"
:1419 INC WR[] "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.INC.WR>,03F]>"
:1420 :
:1421 DEC Q "OPC1/EXTENDED,XDP/<.AND[<SDP/DEC.Q>,03F]>"
:1422 DEC LS[] "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/DEC.LS>,OFF]>,-2]>"
:1423 DEC WR[] "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.DEC.WR>,03F]>"
:1424 :
:1425 COM Q "OPC1/EXTENDED,XDP/<.AND[<SDP/COM.Q>,03F]>"
:1426 COM LS[] "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/COM.LS>,OFF]>,-2]>"
:1427 COM WR[] "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.COM.WR>,03F]>"
:1428 :
:1429 TST WR[] "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.PLUS.0.TO.WR>,03F]>"
:1430 TST Q "OPC1/EXTENDED,XDP/<.AND[<SDP/Q.PLUS.0>,03F]>"
:1431 :
:1432 NOP "OPC/BASIC,D.ADRS/0,B.ADRS/0,DP/<.SHIFT[<.AND[<SDP/Q.CMP.LS>,OFF]>,-2]>"

```

1433 PAGE
1434
1435 MACROS FOR SHIFT OPERATION ON WR[B] AND/OR Q-REGISTER
1436
1437 DURING THE SHIFT OPERATIONS THE ALU CC'S CAN BE LOADED BY
1438 USING THE
1439
1440 DT(SIZE)&SET.ALU.CC
1441
1442 MACRO WITH THE MICROWORD. THE FOLLOWING IS A DESCRIPTION OF THE
1443 ALU CC'S WITH EACH FUNCTION.
1444
1445 ROR WR[] -- ROTATE 32 BIT VALUE ONE POSITION RIGHT.
1446
1447 N - SIGN OF INPUT
1448 Z - SET IF INPUT IS ZERO
1449 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1450 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1451
1452 ROL WR[] -- ROTATE 32 BIT VALUE ONE POSITION LEFT.
1453
1454 N - SIGN OF INPUT
1455 Z - SET IF INPUT IS ZERO
1456 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1457 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1458
1459 ASHR WR[] -- SHIFT 32 BIT VALUE RIGHT ONE POSITION AND SIGN EXTEND.
1460
1461 N - SIGN OF INPUT
1462 Z - SET IF INPUT IS ZERO
1463 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1464 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1465
1466 ASHL WR[] -- SHIFT 32 BIT VALUE LEFT ONE POSITION AND INSERT A ZERO IN BOTTOM BIT.
1467
1468 N - SIGN OF INPUT
1469 Z - SET IF INPUT IS ZERO
1470 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1471 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1472
1473 ASHRC WR[].Q -- SHIFT 64 BIT WR.Q ONE POSITION LEFT AND SIGN EXTEND.
1474
1475 N - SIGN OF INPUT WR[]
1476 Z - SET IF INPUT WR[] IS ZERO
1477 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1478 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1479
1480 RORC WR[].Q -- ROTATE 64 BIT WR.Q ONE POSITION RIGHT.
1481
1482 N - SIGN OF INPUT WR[]
1483 Z - SET IF INPUT WR[] IS ZERO
1484 V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1485 C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1486
1487 ASHLC WR[].Q -- SHIFT 64 BIT WR.Q ONE POSITION LEFT AND INSERT ZERO.


```

1488 :
1489 :
1490 : N - SIGN OF INPUT WR[]
1491 : Z - SET IF INPUT WR[] IS ZERO
1492 : V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1493 : C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1494 :
1495 : ROLC WR[].Q -- ROTATE 64 BIT WR.Q ONE POSITION LEFT.
1496 :
1497 : N - SIGN OF INPUT WR[]
1498 : Z - SET IF INPUT WR[] IS ZERO
1499 : V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1500 : C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
1501 :
1502 : SHL2 WR[] -- SHIFT 32 BIT VALUE TWO POSITIONS LEFT INSERTING ZEROES IN THE BOTTOM BITS.
1503 :
1504 : N - SIGN OF INPUT WR[]+WR[]
1505 : Z - SET IF INPUT WR[]+WR[] IS ZERO
1506 : V - OVERFLOW OF INPUT WR[]+WR[]
1507 : C - CARRY OF INPUT WR[]+WR[]
1508 : ROR WR[] 'OPC1/EXTENDED,B.ADRS/@1,SI/ROT.WR,XDP/<.AND[<SDP/ASHR>,03F]>'
1509 :
1510 : ROL WR[] 'OPC1/EXTENDED,B.ADRS/@1,SI/ROT.WR,XDP/<.AND[<SDP/ASHL>,03F]>'
1511 :
1512 :
1513 : ASHR WR[] 'OPC1/EXTENDED,B.ADRS/@1,SI/ASH.WR,XDP/<.AND[<SDP/ASHR>,03F]>'
1514 :
1515 : ASHL WR[] 'OPC1/EXTENDED,B.ADRS/@1,SI/ASH.WR,XDP/<.AND[<SDP/ASHL>,03F]>'
1516 :
1517 : ASHRC WR[].Q 'OPC1/EXTENDED,B.ADRS/@1,SI/ASH.WR.Q,XDP/<.AND[<SDP/ASHRC>,03F]>'
1518 :
1519 : RORC WR[].Q 'OPC1/EXTENDED,B.ADRS/@1,SI/ROT.WR.Q,XDP/<.AND[<SDP/ASHRC>,03F]>'
1520 :
1521 : ASHLC WR[].Q 'OPC1/EXTENDED,B.ADRS/@1,SI/ASH.WR.Q,XDP/<.AND[<SDP/ASHLC>,03F]>'
1522 :
1523 : ROLC WR[].Q 'OPC1/EXTENDED,B.ADRS/@1,SI/ROT.WR.Q,XDP/<.AND[<SDP/ASHLC>,03F]>'
1524 :
1525 : SHL2 WR[] 'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,SI/2,XDP/<.AND[<SDP/SHL2>,03F]>'
1526 :
1527 : COND.ADD&SHIFT WR[] TO WR[].Q 'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,SI/MUL.SHF,XDP/<.AND[<SDP/COND.ADD&SHIFT>,03F]>'
1528 :
1529 : COND.SUB&SHIFT WR[] FROM WR[].Q 'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,SI/MUL.SHF,XDP/<.AND[<SDP/COND.SUB&SHIFT>,03F]>'
1530 :
1531 : UNSIGNED.MUL WR[] TO WR[].Q 'OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,SI/UMUL.SHF,XDP/<.AND[<SDP/COND.ADD&SHIFT>,03F]>'
1532 : SHR WR[] 'OPC1/EXTENDED,B.ADRS/@1,SI/SHF.WR.Q,XDP/<.AND[<SDP/ASHR>,03F]>'
1533 : SHL WR[] 'ASHL WR[@1]'
1534 : SHRC WR[].Q 'OPC1/EXTENDED,B.ADRS/@1,SI/SHF.WR.Q,XDP/<.AND[<SDP/ASHRC>,03F]>'
1535 : SHLC WR[].Q 'ASHLC WR[].Q'
1536 :
1537 :
1538 : MEMORY REQUEST MACRO
1539 :
1540 : * THIS MACRO IS NOT FOR GENERAL USE!!! *
1541 : * THIS IS ONLY FOR USE IN THE FOLLOWING MACRO *
1542 : MEM.FUNC[] 'M.FUNC2/<.SHIFT[<MEM.FUNC/@1>,-2]>,M.FUNC1/<.AND[<MEM.FUNC/@1>,3]>'
    
```

:1543
:1544 MEM.REQ[] ADRES[] DT[]
:1545 WRITE.MEM LSC[]

'OPC2/MEM.REQ, MEM.FUNC[@1], D.ADRS/@2, MDT/@3'
'OPC1/MOVE, XD.ADRS/@1, MDP/<.SHIFT[<.AND[<SDP/MOV.LS.MEM>, 3F]>, -3]>''

```

:1546 .PAGE
:1547
:1548 .MACROS FOR MISCELLANEOUS OPERATIONS
:1549
:1550     IN THE NEBULA MICROPROGRAMMING LANGUAGE THERE IS A NEED FOR
:1551     VARIOUS UNIQUE FUNCTIONAL OPERATIONS. THESE ARE FOUND IN THE
:1552     MISC INSTRUCTIONS. MOST FUNCTIONS ARE EXECUTED BY INSERTING
:1553     THE REQUEST INSIDE THE MISC[] AND MISC2[] MACROS.
:1554
:1555 MISC []          'OPC2/MISC,MISC.1/@1,MISC.2/NOP,MISC.PORT/MISC''
:1556 MISC2 []        'OPC2/MISC,MISC.1/NOP,MISC.2/@1,MISC.PORT/MISC''
:1557 MISC [] WR[]    'MISC [@1],MISC.WRL/0,MISC.WRR/@2''
:1558 MISC2 [] WR[]   'MISC2 [@1],MISC.WRL/0,MISC.WRR/@2''
:1559
:1560 PORT.CONTROL []  'OPC2/MISC,MISC.PORT/PORT,CNTL.FUNC/@1,PORT.SELECT/IDC,PORT.FUNC/CONTROL''
:1561 PORT.WRITE PORT.ADRS[] WITH WR[] 'OPC2/MISC,MISC.PORT/PORT,R.W.FUNC/@1,PORT.SELECT/IDC,PORT.FUNC/WRITE,MISC.WRL/0,MIS
:1562 PORT.READ PORT.ADRS[] 'OPC2/MISC,MISC.PORT/PORT,R.W.FUNC/@1,PORT.SELECT/IDC,PORT.FUNC/READ''
:1563
:1564     A FEW CASES OF THE MISCELLANEOUS FUNCTIONS HAVE BEEN CALLED OUT AS
:1565     UNIQUE FUNCTIONS IN THE MICROPROGRAMMING LANGUAGE SET. THESE
:1566     ARE LISTED BELOW.
:1567
:1568     THE FOLLOWING TO MACROS SEND DATA TO EXTERNAL ACCELERATORS.
:1569
:1570 ASSERT.DA.AND.PASS.WRO 'MISC2 [ASSERT.DA.AND.PASS.WR] WR[0]''
:1571 ASSERT.DA.AND.PASS.WR1 'MISC2 [ASSERT.DA.AND.PASS.WR] WR[1]''
:1572
:1573     STATE REGISTER MACROS - THE POSSIBLE CHANGES TO THE STATE
:1574     BITS ARE SPECIFIED AS INDIVIDUAL FUNCTIONS.
:1575
:1576 CLR.STATE.ZERO      'OPC2/MISC,MISC.1/CLR.S0,MISC.2/NOP,MISC.PORT/MISC''
:1577 CLR.STATE.ONE       'OPC2/MISC,MISC.1/CLR.S1,MISC.2/NOP,MISC.PORT/MISC''
:1578 SET.STATE.ZERO      'OPC2/MISC,MISC.1/SET.S0,MISC.2/NOP,MISC.PORT/MISC''
:1579 SET.STATE.ONE       'OPC2/MISC,MISC.1/SET.S1,MISC.2/NOP,MISC.PORT/MISC''
:1580 SET.STATE.ZERO&CLR.STATE.ONE 'OPC2/MISC,MISC.1/CLR.S1&SET.S0,MISC.2/NOP,MISC.PORT/MISC''
:1581 SET.STATE.ONE&CLR.STATE.ZERO 'OPC2/MISC,MISC.1/SET.S1&CLR.S0,MISC.2/NOP,MISC.PORT/MISC''
:1582 CLR.STATE.ONE&CLR.STATE.ZERO 'OPC2/MISC,MISC.1/CLR,MISC.2/NOP,MISC.PORT/MISC''
:1583 SET.BACKUP.MASK.VALID 'OPC2/MISC,MISC.1/SET.BACKUP.MASK.VALID,MISC.2/NOP,MISC.PORT/MISC''
:1584
:1585
:1586 ; CONTEXT SIZE AND CONDITION CODE MACROS
:1587
:1588 DT(SIZE)&MAP.ALU.CC.TO.PSL 'CC/DT(SIZE)&MAP.ALU.CC.TO.PSL''
:1589 DT(LONG)&SET.ALU.CC       'CC/DT(LONG)&SET.ALU.CC''
:1590 DT(SIZE)&SET.ALU.CC       'CC/DT(SIZE)&SET.ALU.CC''
:1591
    
```



```

:1592 PAGE
:1593 JUMP AND MICRO SEQUENCER CONTROL MACROS
:1594
:1595 THE FOLLOWING MACROS ARE USED TO CONTROL THE PROGRAMS FLOW OF
:1596 CONTROL.
:1597
:1598 JMP [] -- TRANSFER TO NEW ADDRESS UNCONDITIONALLY.
:1599
:1600 JMP.OS [] -- TRANSFER TO RESULTANT ADDRESS OF LOGICAL OR OF ADDRESS
:1601 IN THE INSTRUCTION AND THE OS REGISTER UNCONDITIONALLY.
:1602
:1603 JMP.IF[] TO [] -- TRANSFER TO NEW ADDRESS ONLY IF JUMP
:1604 CONDITION IS MET.
:1605
:1606 JSR [] -- TRANSFER TO NEW ADDRESS UNCONDITIONALLY AND PUT
:1607 RETURN ADDRESS ON THE SEQUENCER STACK.
:1608
:1609 JSR.OS [] -- TRANSFER TO RESULTANT ADDRESS OF LOGICAL OR OF
:1610 ADDRESS IN THE INSTRUCTION AND THE OS REGISTER.
:1611 THE RETURN ADDRESS IS ALSO PUSHED ON THE SEQUENCER STACK.
:1612
:1613 RETURN -- TRANSFER TO ADDRESS ON TOP OF THE SEQUENCER STACK AND
:1614 POP THE STACK.
:1615
:1616 RETURN+1 -- TRANSFER TO ADDRESS ON TOP OF THE SEQUENCER
:1617 STACK PLUS ONE AND POP THE STACK.
:1618
:1619 RETURN+1.IF(MEM.REF.OK) -- IF THE MOST RECENT MEMORY REQUEST HAD
:1620 NO ERROR THEN TRANSFER TO ADDRESS ON TOP OF THE SEQUENCER
:1621 STACK PLUS ONE AND POP THE STACK. IF THERE WAS AN
:1622 ERROR THEN SKIP OVER THE ENEXT INSTRUCTION.
:1623
:1624 LOOP.IF(NEQ) -- IF THE ALU Z-BIT IS ZERO (LAST VALUE CALCULATED
:1625 WHEN THE ALU.CC'S WERE SET DID NOT RESULT AS ZERO)
:1626 THEN TRANSFER TO THE ADDRESS ON TOP OF THE SEQUENCER
:1627 STACK. THE STACK IS NOT POPPED. IF THE ALU Z-BIT
:1628 IS ONE THEN CONTINUE WITH THE NEXT INSTRUCTION.
:1629 (UPC+1) AND THE STACK IS POPPED.
:1630
:1631 LOOP.IF(GEQU) -- IF THE ALU C-BIT IS ONE (LAST VALUE CALCULATED WHEN
:1632 THE ALU.CC'S WERE SET WAS GREATER THAN OR EQUAL TO ZERO)
:1633 THEN TRANSFER TO THE ADDRESS ON TOP OF THE SEQUENCER
:1634 STACK. THE STACK IS NOT POPPED. IF THE ALU C-BIT
:1635 IS ZERO THEN CONTINUE WITH THE NEXT INSTRUCTION
:1636 (UPC+1) AND THE STACK IS POPPED.
:1637
:1638 LOOP.IF(NO INTERRUPT AND NO SYNC) -- IF THE INTERRUPT CONDITION
:1639 IS NOT TRUE AND THE ACCELERATOR SYNC CONDITION IS NOT
:1640 TRUE THEN TRANSFER TO THE ADDRESS ON THE TOP OF THE SEQUENCER
:1641 STACK. THE STACK IS NOT POPPED. IF THE INTERRUPT
:1642 CONDITION IS TRUE THEN CONTINUE WITH THE NEXT INSTRUCTION
:1643 (UPC+1) AND THE SEQUENCER STACK IS NOT POPPED. IF THE ACCELERATOR
:1644 SYNC IS TRUE THEN CONTINUE WITH THE NEXT INSTRUCTION
:1645 (UPC+1) AND THE SEQUENCER STACK IS POPPED.
:1646

```

```

:1647 : SKIP.IF[] -- IF THE SKIP CONDITION IS SATISFIED THEN TRANSFER
:1648 : TO UPC+2 ELSE TRANSFER TO UPC+1.
:1649 :
:1650 :
:1651 JMP [] "OPC2/JUMP,JUMP.ADRS/@1,JCTL/JUMP"
:1652 JMP.OS [] "OPC2/JUMP,JUMP.ADRS/@1,OS/WITH.OS,JCTL/JUMP"
:1653 JMP.IF[] TO [] "OPC2/JUMP,JCTL/@1,JUMP.ADRS/@2"
:1654 JSR [] "OPC2/JUMP,JCTL/JSR,JUMP.ADRS/@1"
:1655 JSR.OS [] "OPC2/JUMP,JCTL/JSR,JUMP.ADRS/@1,OS/WITH.OS"
:1656 :
:1657 RETURN "SCTL/RETURN"
:1658 RETURN+1 "SCTL/RETURN+1"
:1659 RETURN+1.IF(MEM.REF.OK) "SCTL/RET.NOERR"
:1660 :
:1661 LOOP.IF(NEQ) "SCTL/JMP.Z.CLR"
:1662 LOOP.IF(GEQU) "SCTL/JMP.C.SET"
:1663 LOOP.IF(NO INTERRUPT AND NO SYNC) ELSE SKIP.IF(SYNC) "SCTL/LOOP.IF.NO.I.AND.SYNC"
:1664 :
:1665 SKIP.IF[] "SCTL/@1"
:1666 SKIP "SCTL/SKIP"
  
```

```
:1667 .PAGE
:1668 :
:1669 : INSTRUCTION STREAM MACROS
:1670 :
:1671 : * THIS MACRO IS NOT FOR GENERAL USE!!! *
:1672 : * THIS IS ONLY FOR USE IN THE FOLLOWING MACROS *
:1673 DEC[] "OPC2/DECODE,DEC.ADRS/<.SHIFT[<JUMP.ADRS/@1>,-8]>"
:1674
:1675 DECODE.SPEC ADRS[] "DEC[@1],IFUNC/SPEC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/S
:1676 DECODE.ASRC ADRS[] "DEC[@1],IFUNC/ASRC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/A
:1677 DECODE.ESRC ADRS[] "DEC[@1],IFUNC/ESRC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/E
:1678 DECODE.VSRC ADRS[] "DEC[@1],IFUNC/VSRC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/V
:1679 DECODE.FLOAT ADRS[] "DEC[@1],IFUNC/FLOAT,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/
:1680 DECODE.OPC1 ADRS[] "DEC[@1],IFUNC/VAX.IRD,R.DST/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,OPC.SPEC/VAX.IRD"
:1681 EXEC.DISPATCH ADRS[] "DEC[@1],IFUNC/VAX.EXEC,IB.REQ/NOP,R.DST/NOP,CM.HI/LOW.BYTE,JCTL/JSR,OPC.SPEC/VAX.EXEC"
:1682
:1683 DECODE.CM.OPC ADRS[] "DEC[@1],IFUNC/CM.IRD,CM.HI/HI.BYTE,OPC.SPEC/CM.IRD,LD.OS/LOAD.OS"
:1684 DECODE.CM.DST ADRS[] "DEC[@1],IFUNC/CM.DST,OPC.SPEC/CM.DST,LD.OS/LOAD.OS,R.DST/ENAB"
:1685 DECODE.CM.SINGLE ADRS[] "DEC[@1],IFUNC/CM.SINGLE,OPC.SPEC/CM.SINGLE"
:1686 CM.EXEC ADRS[] "DEC[@1],IFUNC/CM.EXEC,JCTL/JUMP,OPC.SPEC/CM.EXEC,LD.OS/LOAD.OS"
:1687
:1688 SAVE.PC WRO "BPC/SAVE.PC"
:1689 SAVE.PC WR4 "BPC/SAVE.PC"
:1690 SAVE.CM.PC WRO "BPC/SAVE.PC"
:1691 SAVE.CM.PC WR4 "BPC/SAVE.PC"
:1692
:1693 :
:1694 : THESE SKIP CONDITIONS ARE USED FOR THE DECODE MICRO INSTRUCTION
:1695 :
:1696 SKIP.IF(1B VALID) "JCTL/NO.JUMP.TST"
:1697 JMP.IF(1B VALID) "JCTL/JUMP.VALID"
:1698 JSR.IF(1B VALID) "JCTL/JSR.VALID"
:1699 :
:1700 : THESE SKIP CONDITIONS ARE TO BE USED WITH THE COMPATIBILITY MODE DECODE
:1701 : MACROS.
:1702 :
:1703 JMP.TO [] "JCTL/JUMP,DEC[@1]"
:1704 JSR.TO [] "JCTL/JSR,DEC[@1]"
:1705 .BIN
```



```

:1706 .PAGE
:1707 .TOC "FIELD DEFINITIONS FOR DATA PATH CONTROL MICRO WORD VER 00.01"
:1708
:1709 : THIS SECTION IS FOR THE DATA PATH CONTROL ROMS
:1710 :
:1711
:1712 .CCODE
:1713 SDP/=<8:0>,.ADDRESS,.VALIDITY=0
:1714
:1715 : THIS FIELD CORRESPONDS TO THE 2901 ALU FUNCTION CONTROL PINS ... 15,14,13
:1716
:1717 ALU/=<2:0>,.DEFAULT=<ALU/R.OR.S>
:1718
:1719 R.PLUS.S=0 : ADD R WITH S
:1720 S.MINUS.R=1 : SUBTRACT R FROM S
:1721 R.MINUS.S=2 : SUBTRACT S FROM R
:1722 R.OR.S=3 : OR R WITH S
:1723 R.AND.S=4 : AND R WITH S
:1724 NOTR.AND.S=5 : COMPLEMENT R THEN AND WITH S
:1725 R.XOR.S=6 : XOR R WITH S
:1726 R.XNOR.S=7 : XOR R WITH S THEN COMPLEMENT THE RESULT
:1727 SRC/=<8:6>,.DEFAULT=<SRC/D.0>
:1728
:1729 O.Q=2 : RMX=0 SMX=Q
:1730 O.B=3 : RMX=0 SMX=B
:1731 A.Q=0 : RMX=A SMX=Q
:1732 A.B=01 : RMX=A SMX=B
:1733 D.Q=06 : RMX=D SMX=Q
:1734 D.O=7 : RMX=D SMX=Q
:1735 O.A=4 : RMX=0 SMX=A
:1736 D.A=5 : RMX=D SMX=A
:1737
:1738 : THIS FIELD CORRESPONDES TO THE 2901 ALU DESTINATION
:1739 : CONTROL PINS 18, 17 AND 16.
:1740
:1741 DST/=<5:3>,.DEFAULT=<DST/NOP>
:1742
:1743 LOAD.Q=0 : LOAD Q-REGISTER
:1744 NOP=1 : HOLD ALL REGISTERS
:1745 WRITE.B.A=2 : WRITE RAM B-PORT AND OUTPUT RAM A-PORT
:1746 WRITE.B.F=3 : WRITE RAM B-PORT AND OUTPUT ALU
:1747 RSHF.RAM.Q=4 : RIGHT SHIFT RAM AND Q
:1748 RSHF.RAM=5 : RIGHT SHIFT RAM
:1749 LSHF.RAM.Q=6 : LEFT SHIFT RAM AND Q
:1750 LSHF.RAM=7 : LEFT SHIFT RAM
:1751
:1752 CIN/=<9>,.DEFAULT=0
:1753
:1754 NO.CIN=0 : NO CARRY IN TO ALU
:1755 CIN=1 : CARRY IN TO ALU
:1756
:1757
:1758

```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

FIELD DEFINITIONS 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
FIELD DEFINITIONS FOR DATA PATH CONTROL MICRO WORD

Fiche 1 Frame K4
ENKCA Micro-diagnostic for DAP module
VER 00.01

Sequence 49
Rev 01.00

Page 43

```
:1759 .PAGE
:1760 .TOC "CONTROL ROM CONTENTS VER 00.01"
:1761 .SEQUENTIAL
:1762 : THESE ARE THE DATA PATH CONTROL ROM CONTENTS
:1763 :
:1764 : THE FOLLOWING LOCATIONS ARE USED BY THE DECODE.IS MICRO INSTRUCTION
:1765 :
:1766 : THIS INSTRUCTION WILL DO PC+1 OPERATIONS
:1767 :
:1768 : BACK UP PC IN 2901 RAM
:1769 000:
:1770 DECODE.IS:
C 0000. 3D8 :1771 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0001. 3D8 :1772 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0002. 3D8 :1773 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0003. 3D8 :1774 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0004. 3D8 :1775 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0005. 3D8 :1776 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0006. 3D8 :1777 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0007. 3D8 :1778 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0008. 3D8 :1779 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0009. 3D8 :1780 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000A. 3D8 :1781 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000B. 3D8 :1782 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000C. 3D8 :1783 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000D. 3D8 :1784 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000E. 3D8 :1785 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000F. 3D8 :1786 SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
:1787 :
:1788 :
:1789 : DON'T BACK UP PC
C 0010. 3C8 :1790 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0011. 3C8 :1791 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0012. 3C8 :1792 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0013. 3C8 :1793 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0014. 3C8 :1794 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0015. 3C8 :1795 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0016. 3C8 :1796 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0017. 3C8 :1797 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0018. 3C8 :1798 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0019. 3C8 :1799 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001A. 3C8 :1800 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001B. 3C8 :1801 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001C. 3C8 :1802 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001D. 3C8 :1803 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001E. 3C8 :1804 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001F. 3C8 :1805 SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) CONTROL ROM CONTENT 3-MAR-82
CONTROL ROM CONTENTS

Fiche 1 Frame L4
ENKCA Micro-diagnostic for DAP module
VER 00.01

Sequence 50
Rev 01.00
Page 44

```
:1806 .PAGE
:1807 : THESE ADDRESSES ARE USED BY THE 'JUMP' MICRO INSTRUCTION
:1808 :
:1809 : THE FOLLOWING CONTROL CODES WILL INSURE THAT NO DATA IS DESTROYED WITHIN
:1810 : THE 2901 DURING JUMP OR JSR OPERATIONS.
:1811
:1812 020:
:1813 JUMP:
C 0020. 3CB :1814 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0021. 3CB :1815 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0022. 3CB :1816 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0023. 3CB :1817 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0024. 3CB :1818 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0025. 3CB :1819 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0026. 3CB :1820 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0027. 3CB :1821 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0028. 3CB :1822 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0029. 3CB :1823 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002A. 3CB :1824 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002B. 3CB :1825 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002C. 3CB :1826 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002D. 3CB :1827 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002E. 3CB :1828 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002F. 3CB :1829 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0030. 3CB :1830 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0031. 3CB :1831 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0032. 3CB :1832 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0033. 3CB :1833 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0034. 3CB :1834 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0035. 3CB :1835 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0036. 3CB :1836 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0037. 3CB :1837 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0038. 3CB :1838 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0039. 3CB :1839 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003A. 3CB :1840 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003B. 3CB :1841 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003C. 3CB :1842 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003D. 3CB :1843 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003E. 3CB :1844 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003F. 3CB :1845 SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

M 4
CONTROL ROM CONTENT 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
CONTROL ROM CONTENTS

Fiche 1 Frame M4
ENKCA Micro-diagnostic for DAP module
VER 00.01

Sequence 51
Rev 01.00

Page 45

:1846 :PAGE
:1847 : THESE LOCATIONS ARE USED BY THE MISC' MICRO INSTRUCTION
:1848 :
:1849 : THE FOLLOWING CONTROL CODES WILL INSURE THAT NO DATA IS DESTROYED WITHIN
:1850 : THE 2901 DURING MISC INSTRUCTION EXECUTION.
:1851 :
:1852 :
:1853 :
:1854 :
:1855 :
:1856 :
:1857 :
:1858 :
:1859 :
:1860 :
:1861 :
:1862 :
:1863 :
:1864 :
:1865 :
:1866 :
:1867 :
:1868 :
:1869 :
:1870 :
:1871 :
:1872 :
:1873 :
:1874 :
:1875 :
:1876 :
:1877 :
:1878 :
:1879 :
:1880 :
:1881 :
:1882 :
:1883 :
:1884 :
:1885 :

040:
MISC:

C 0040.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0041.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0042.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0043.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0044.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0045.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0046.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0047.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0048.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0049.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004A.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004B.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004C.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004D.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004E.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004F.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0050.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0051.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0052.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0053.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0054.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0055.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0056.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0057.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0058.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0059.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005A.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005B.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005C.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005D.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005E.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005F.	313	SRC/O.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) CONTROL ROM CONTENT 3-MAR-82
CONTROL ROM CONTENTS

Fiche 1 Frame N4

Sequence 52
Rev 01.00

Page 46

VER 00.01

```
:1886 .PAGE
:1887 : THESE ADDRESSES ARE USED BY THE MEM.REQ MICRO INSTRUCTION
:1888 :
:1889 : THIS INSTRUCTION CAUSES WR[5] TO BE LOADED WITH THE VIRTUAL ADDRESS
:1890 : OF THE MEMORY REFERENCE.
:1891 060:
:1892 MEM.REQ:
C 0060, 3DB :1893 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0061, 3DB :1894 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0062, 3DB :1895 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0063, 3DB :1896 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0064, 3DB :1897 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0065, 3DB :1898 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0066, 3DB :1899 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0067, 3DB :1900 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0068, 3DB :1901 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0069, 3DB :1902 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006A, 3DB :1903 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006B, 3DB :1904 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006C, 3DB :1905 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006D, 3DB :1906 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006E, 3DB :1907 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006F, 3DB :1908 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0070, 3DB :1909 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0071, 3DB :1910 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0072, 3DB :1911 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0073, 3DB :1912 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0074, 3DB :1913 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0075, 3DB :1914 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0076, 3DB :1915 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0077, 3DB :1916 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0078, 3DB :1917 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0079, 3DB :1918 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007A, 3DB :1919 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007B, 3DB :1920 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007C, 3DB :1921 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007D, 3DB :1922 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007E, 3DB :1923 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007F, 3DB :1924 SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
```

```

:1925 .PAGE
:1926 : THESE ARE THE ADDRESSES FOR THE EXTENDED MICRO INSTRUCTION
:1927 : ADDRESSES START AT 80-FF
:1928 080:
:1929
:1930 WR.PLUS.O.TO.WR:
C 0080, 118 :1931 SRC/O.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
:1932
:1933 WR.INC.WR:
C 0081, 318 :1934 SRC/O.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
:1935
:1936 WR.NEG.WR:
C 0082, 31A :1937 SRC/O.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
:1938
:1939 WR.COM.WR:
C 0083, 31F :1940 SRC/O.A,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
:1941
:1942 WR.DEC.WR:
C 0084, 119 :1943 SRC/O.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
:1944
:1945 FILL85:
C 0085, 10B :1946 SRC/O.A
:1947
:1948 MOV.Q.WR:
C 0086, 29B :1949 SRC/O.Q,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
:1950
:1951 FILL87:
C 0087, 08B :1952 SRC/O.Q
:1953
:1954 COND.ADD&SHIFT:
C 0088, 060 :1955 SRC/A.B,ALU/R.PLUS.S,DST/RSHF.RAM.Q,CIN/NO.CIN
:1956
:1957 COND.SUB&SHIFT:
C 0089, 261 :1958 SRC/A.B,ALU/S.MINUS.R,DST/RSHF.RAM.Q,CIN/CIN
:1959
:1960 FILL8A:
C 008A, 04B :1961 SRC/A.B
:1962
:1963 SHL2:
C 008B, 078 :1964 SRC/A.B,ALU/R.PLUS.S,DST/LSHF.RAM,CIN/NO.CIN
:1965
:1966 ASHRC:
C 008C, 0E3 :1967 SRC/O.B,ALU/R.OR.S,DST/RSHF.RAM.Q,CIN/NO.CIN
:1968
:1969 ASHR:
C 008D, 2EB :1970 SRC/O.B,ALU/R.OR.S,DST/RSHF.RAM,CIN/CIN
:1971
:1972 ASHLC:
C 008E, 2F3 :1973 SRC/O.B,ALU/R.OR.S,DST/LSHF.RAM.Q,CIN/CIN
:1974
:1975 ASHL:
C 008F, 2FB :1976 SRC/O.B,ALU/R.OR.S,DST/LSHF.RAM,CIN/CIN
:1977
:1978 Q.PLUS.O:
C 0090, 088 :1979 SRC/O.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
:1980
:1981 FILL91:
C 0091, 08B :1982 SRC/O.Q
:1983
:1984 WR.CMP.Q:
C 0092, 20A :1985 SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
:1986
:1987 Q.CMP.WR:
C 0093, 209 :1988 SRC/A.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
:1989
:1990 DEC.Q:
C 0094, 081 :1991 SRC/O.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/NO.CIN
:1992
:1993 INC.Q:
C 0095, 280 :1994 SRC/O.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/CIN
:1995
:1996 NEG.Q:

```


ZZ-ENKCA-1.0	:	ENKCA.MIC	MICRO2 1L(03)	CONTROL ROM CONTENT 3-MAR-82 09:34:33	C 5	Fiche 1 Frame C5	Sequence 54	Page 48
: ENKCA.MCR				ENKCA Micro-diagnostic for DAP module			Rev 01.00	
: ENKCA.MIC			CONTROL ROM CONTENTS			VER 00.01		

C 0096, 282	:1980	SRC/O.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
	:1981	COM.Q:
C 0097, 287	:1982	SRC/O.Q,ALU/R.XNOR.S,DST/LOAD.Q,CIN/CIN
	:1983	
	:1984	WR.BIT.WR:
C 0098, 04C	:1985	SRC/A.B,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
	:1986	WR.CMP.WR:
C 0099, 24A	:1987	SRC/A.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
	:1988	FILL9A:
C 009A, 04B	:1989	SRC/A.B
	:1990	WR.PLUS.WR+1:
C 009B, 258	:1991	SRC/A.B,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
	:1992	
	:1993	FILL9C:
C 009C, 04B	:1994	SRC/A.B
	:1995	FILL9D:
C 009D, 04B	:1996	SRC/A.B
	:1997	FILL9E:
C 009E, 0CB	:1998	SRC/O.B
	:1999	FILL9F:
C 009F, 0CB	:2000	SRC/O.B
	:2001	
	:2002	WR.PLUS.Q:
C 00A0, 000	:2003	SRC/A.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
	:2004	WR.FROM.Q:
C 00A1, 201	:2005	SRC/A.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
	:2006	Q.FROM.WR.Q:
C 00A2, 202	:2007	SRC/A.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
	:2008	WR.OR.Q:
C 00A3, 203	:2009	SRC/A.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
	:2010	
	:2011	WR.AND.Q:
C 00A4, 004	:2012	SRC/A.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
	:2013	WR.MASK.Q:
C 00A5, 205	:2014	SRC/A.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/CIN
	:2015	WR.XOR.Q:
C 00A6, 206	:2016	SRC/A.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
	:2017	FILLA7:
C 00A7, 00B	:2018	SRC/A.Q
	:2019	
	:2020	WR.PLUS.WR.Q:
C 00A8, 040	:2021	SRC/A.B,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
	:2022	WR.FROM.WR.Q:
C 00A9, 241	:2023	SRC/A.B,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
	:2024	FILLAA:
C 00AA, 04B	:2025	SRC/A.B
	:2026	WR.OR.WR.Q:
C 00AB, 243	:2027	SRC/A.B,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
	:2028	
	:2029	WR.AND.WR.Q:
C 00AC, 044	:2030	SRC/A.B,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
	:2031	WR.MASK.WR.Q:
C 00AD, 245	:2032	SRC/A.B,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/CIN
	:2033	WR.XOR.WR.Q:
C 00AE, 246	:2034	SRC/A.B,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03)

CONTROL ROM CONTENTS

CONTROL ROM CONTENT 3-MAR-82 09:34:33

D 5

ENKCA Micro-diagnostic for DAP module

Fiche 1 Frame D5

Sequence 55

Rev 01.00

Page 49

VER 00.01

C 00AF, 04B :2035 FILLAF:
:2036 SRC/A.B
:2037
:2038 Q.PLUS.WR:
:2039 SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
:2040 WR.FROM.Q.WR:
:2041 SRC/A.Q,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
:2042 Q.FROM.WR:
:2043 SRC/A.Q,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
:2044 Q.OR.WR:
:2045 SRC/A.Q,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
:2046
:2047 Q.AND.WR:
:2048 SRC/A.Q,ALU/R.AND.S,DST/WRITE.B.F,CIN/NO.CIN
:2049 FILLB5:
:2050 SRC/A.Q
:2051 Q.XOR.WR:
:2052 SRC/A.Q,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
:2053 Q.BIT.WR:
:2054 SRC/A.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
:2055
:2056 WR.PLUS.WR:
:2057 SRC/A.B,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
:2058 WR.FROM.WR:
:2059 SRC/A.B,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
:2060 WR.FROM.WR.WR:
:2061 SRC/A.B,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
:2062 WR.OR.WR:
:2063 SRC/A.B,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
:2064
:2065 WR.FROM.WR-1:
:2066 SRC/A.B,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
:2067 WR.MASK.WR:
:2068 SRC/A.B,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
:2069 WR.XOR.WR:
:2070 SRC/A.B,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
:2071 WR.AND.WR:
:2072 SRC/A.B,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
:2073

```

:2074 .PAGE
:2075 : THIS IS THE DP CODES FOR THE 'MOVE' MICRO INSTRUCTION
:2076
:2077 0C0:
:2078
:2079 MOV.MEM.WR:
C 00C0, 1D3 :2080 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C1, 1D3 :2081 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C2, 1D3 :2082 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C3, 1D3 :2083 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C4, 1D3 :2084 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C5, 1D3 :2085 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C6, 1D3 :2086 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C7, 1D3 :2087 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
:2088
:2089 MOV.LS.MEM:
C 00C8, 1CB :2090 SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00C9, 1CB :2091 SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CA, 1CB :2092 SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CB, 1CB :2093 SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CC, 1CB :2094 SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CD, 1CB :2095 SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CE, 1CB :2096 SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CF, 1CB :2097 SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
:2098
:2099 MOV.XWR.Q:
C 00D0, 0C3 :2100 SRC/O.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D1, 0C3 :2101 SRC/O.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D2, 0C3 :2102 SRC/O.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D3, 0C3 :2103 SRC/O.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D4, 0C3 :2104 SRC/O.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D5, 0C3 :2105 SRC/O.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D6, 0C3 :2106 SRC/O.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D7, 0C3 :2107 SRC/O.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
:2108
:2109 MOV.LS.WR:
C 00D8, 1DB :2110 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00D9, 1DB :2111 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DA, 1DB :2112 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DB, 1DB :2113 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DC, 1DB :2114 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DD, 1DB :2115 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DE, 1DB :2116 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DF, 1DB :2117 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
:2118
  
```


	:2119	.PAGE	
	:2120	MOV.MEM.LS:	
C 00E0, 1CB	:2121		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E1, 1CB	:2122		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E2, 1CB	:2123		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E3, 1CB	:2124		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E4, 1CB	:2125		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E5, 1CB	:2126		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E6, 1CB	:2127		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E7, 1CB	:2128		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
	:2129	MOV.ACC.LS:	
C 00E8, 1CB	:2130		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E9, 1CB	:2131		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00EA, 1CB	:2132		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00EB, 1CB	:2133		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00EC, 1CB	:2134		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00ED, 1CB	:2135		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00EE, 1CB	:2136		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00EF, 1CB	:2137		SRC/D.O,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
	:2138		
	:2139		
	:2140	WR.PLUS.OS:	
C 00F0, 148	:2141		SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F1, 148	:2142		SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F2, 148	:2143		SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F3, 148	:2144		SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F4, 148	:2145		SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F5, 148	:2146		SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F6, 148	:2147		SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F7, 148	:2148		SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
	:2149		
	:2150	MOV.WR.LS:	
C 00F8, 10B	:2151		SRC/O.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00F9, 10B	:2152		SRC/O.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FA, 10B	:2153		SRC/O.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FB, 10B	:2154		SRC/O.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FC, 10B	:2155		SRC/O.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FD, 10B	:2156		SRC/O.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FE, 10B	:2157		SRC/O.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FF, 10B	:2158		SRC/O.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
	:2159		

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) CONTROL ROM CONTENTS
3-MAR-82 09:34:33

CONTROL ROM CONTENT 3-MAR-82

G 5

ENKCA Micro-diagnostic for DAP module
VER 00.01

Fiche 1 Frame G5

Sequence 58
Rev 01.00

Page 52

```
:2160 .PAGE
:2161 : THIS GROUP OF INSTRUCTIONS IS FOR THE 'BASIC' MICRO INSTRUCTION
:2162 : THIS USES ADDRESSES 100-1FF
:2163
:2164 100:
:2165 LS.PLUS.WR:
C 0100, 158 :2166 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0101, 158 :2167 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0102, 158 :2168 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0103, 158 :2169 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
:2170
:2171 LS.FROM.WR:
C 0104, 359 :2171 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
C 0105, 359 :2172 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
C 0106, 359 :2173 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
C 0107, 359 :2174 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
:2175
:2176 LS.AND.WR:
C 0108, 35C :2176 SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
C 0109, 35C :2177 SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
C 010A, 35C :2178 SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
C 010B, 35C :2179 SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
:2180
:2181 LS.XOR.WR:
C 010C, 35E :2181 SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
C 010D, 35E :2182 SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
C 010E, 35E :2183 SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
C 010F, 35E :2184 SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
:2185
:2186 LS.FROM.WR-1:
C 0110, 159 :2187 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
C 0111, 159 :2188 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
C 0112, 159 :2189 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
C 0113, 159 :2190 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
:2191
:2192 LS.MASK.WR:
C 0114, 35D :2192 SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
C 0115, 35D :2193 SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
C 0116, 35D :2194 SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
C 0117, 35D :2195 SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
:2196
:2197 LS.PLUS.WR+1:
C 0118, 358 :2197 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0119, 358 :2198 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 011A, 358 :2199 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 011B, 358 :2200 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
:2201
:2202 LS.OR.WR:
C 011C, 35B :2202 SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 011D, 35B :2203 SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 011E, 35B :2204 SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 011F, 35B :2205 SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
:2206
:2207 WR.PLUS.LS.Q:
C 0120, 140 :2208 SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0121, 140 :2209 SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0122, 140 :2210 SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0123, 140 :2211 SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
:2212
:2213 LS.FROM.WR.Q:
C 0124, 341 :2213 SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
C 0125, 341 :2214 SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
```


C 0126, 341	:2215	SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
C 0127, 341	:2216	SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
	:2217	WR.FROM.LS.Q:
C 0128, 342	:2218	SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 0129, 342	:2219	SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 012A, 342	:2220	SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 012B, 342	:2221	SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
	:2222	WR.OR.LS.Q:
C 012C, 343	:2223	SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 012D, 343	:2224	SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 012E, 343	:2225	SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 012F, 343	:2226	SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
	:2227	
	:2228	WR.AND.LS.Q:
C 0130, 144	:2229	SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0131, 144	:2230	SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0132, 144	:2231	SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0133, 144	:2232	SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
	:2233	WR.XOR.LS.Q:
C 0134, 346	:2234	SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0135, 346	:2235	SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0136, 346	:2236	SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0137, 346	:2237	SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
	:2238	LS.CMP.WR:
C 0138, 34A	:2239	SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 0139, 34A	:2240	SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 013A, 34A	:2241	SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 013B, 34A	:2242	SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
	:2243	WR.CMP.LS:
C 013C, 349	:2244	SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 013D, 349	:2245	SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 013E, 349	:2246	SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 013F, 349	:2247	SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
	:2248	
	:2249	LS.PLUS.Q:
C 0140, 180	:2250	SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0141, 180	:2251	SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0142, 180	:2252	SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0143, 180	:2253	SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
	:2254	LS.FROM.Q:
C 0144, 381	:2255	SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
C 0145, 381	:2256	SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
C 0146, 381	:2257	SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
C 0147, 381	:2258	SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
	:2259	Q.FROM.LS.Q:
C 0148, 382	:2260	SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 0149, 382	:2261	SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 014A, 382	:2262	SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 014B, 382	:2263	SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
	:2264	LS.OR.Q:
C 014C, 383	:2265	SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 014D, 383	:2266	SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 014E, 383	:2267	SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 014F, 383	:2268	SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
	:2269	

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) CONTROL ROM CONTENT 3-MAR-82
3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
CONTROL ROM CONTENTS

Fiche 1 Frame 15

Sequence 60

Rev 01.00

Page 54

VER 00.01

```
:2270 LS.MASK.Q:
C 0150, 185 :2271 SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0151, 185 :2272 SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0152, 185 :2273 SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0153, 185 :2274 SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
:2275 LS.XOR.Q:
C 0154, 386 :2276 SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0155, 386 :2277 SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0156, 386 :2278 SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0157, 386 :2279 SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
:2280 LS.AND.Q:
C 0158, 384 :2281 SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
C 0159, 384 :2282 SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
C 015A, 384 :2283 SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
C 015B, 384 :2284 SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
:2285 LS.BIT.Q:
C 015C, 38C :2286 SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
C 015D, 38C :2287 SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
C 015E, 38C :2288 SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
C 015F, 38C :2289 SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
:2290 MOV.LS.Q:
C 0160, 1C3 :2291 SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 0161, 1C3 :2292 SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 0162, 1C3 :2293 SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 0163, 1C3 :2294 SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
:2295 WR.BIT.LS:
C 0164, 34C :2296 SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0165, 34C :2297 SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0166, 34C :2298 SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0167, 34C :2299 SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
:2300 LS.CMP.Q:
C 0168, 38A :2301 SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 0169, 38A :2302 SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 016A, 38A :2303 SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 016B, 38A :2304 SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
:2305 Q.CMP.LS:
C 016C, 389 :2306 SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 016D, 389 :2307 SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 016E, 389 :2308 SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 016F, 389 :2309 SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
:2310 Q.PLUS.LS.TO.WR:
C 0170, 198 :2311 SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0171, 198 :2312 SRC/L.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0172, 198 :2313 SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0173, 198 :2314 SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
:2315 WRA.FROM.LS.WRB:
C 0174, 35A :2316 SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
C 0175, 35A :2317 SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
C 0176, 35A :2318 SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
C 0177, 35A :2319 SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
:2320 LS.NEG.WR:
C 0178, 3D9 :2321 SRC/D.Q,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
C 0179, 3D9 :2322 SRC/D.Q,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
:2323
```

ZZ-ENKCA-1.0	:	ENKCA.MIC								
:	:	ENKCA.MCR								
:	:	ENKCA.MIC								
			MICRO2	1L(03)	3-MAR-82	09:34:33	J 5	Fiche 1	Frame J5	Sequence 61
			CONTROL ROM CONTENTS					ENKCA Micro-diagnostic for DAP module	Rev 01.00	Page 55
								VER 00.01		
C 017A, 3D9	:	2325								
C 017B, 3D9	:	2326								
	:	2327								
C 017C, 3DF	:	2328								
C 017D, 3DF	:	2329								
C 017E, 3DF	:	2330								
C 017F, 3DF	:	2331								

		SRC/D.0,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
		SRC/D.0,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
LS.COM.WR:		
		SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
		SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
		SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
		SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03)
CONTROL ROM CONTENTS

CONTROL ROM CONTENT 3-MAR-82
3-MAR-82 09:34:33

K 5
ENKCA Micro-diagnostic for DAP module
VER 00.01

Fiche 1 Frame K5

Sequence 62
Rev 01.00

Page 56

```
:2332 .PAGE
:2333 :
:2334 : THE FOLLOWING LOCATIONS HAVE THE LOCAL STORE AS THEIR DESTINATION.
:2335 :
:2336 : THIS FACT IS USED BY HARDWARE TO GENERATE LOCAL STORE WRITE PULSES
:2337 :
:2338 180:
:2339
:2340 LS.PLUS.WR.XCHG:
C 0180, 150 :2341 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 0181, 150 :2342 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 0182, 150 :2343 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 0183, 150 :2344 SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
:2345
:2346 LS.FROM.WR.XCHG:
C 0184, 351 :2346 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
C 0185, 351 :2347 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
C 0186, 351 :2348 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
C 0187, 351 :2349 SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
:2350
:2351 LS.AND.WR.XCHG:
C 0188, 354 :2351 SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
C 0189, 354 :2352 SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
C 018A, 354 :2353 SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
C 018B, 354 :2354 SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
:2355
:2356 LS.XOR.WR.XCHG:
C 018C, 356 :2356 SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
C 018D, 356 :2357 SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
C 018E, 356 :2358 SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
C 018F, 356 :2359 SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
:2360
:2361 SWAP.LS.WR:
C 0190, 1D3 :2362 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 0191, 1D3 :2363 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 0192, 1D3 :2364 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 0193, 1D3 :2365 SRC/D.O,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
:2366
:2367 CLR.LS:
C 0194, 3CC :2367 SRC/D.O,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0195, 3CC :2368 SRC/D.O,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0196, 3CC :2369 SRC/D.O,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0197, 3CC :2370 SRC/D.O,ALU/R.AND.S,DST/NOP,CIN/CIN
:2371
:2372 MOV.Q.WR&WR.LS:
C 0198, 293 :2372 SRC/O.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0199, 293 :2373 SRC/O.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 019A, 293 :2374 SRC/O.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 019B, 293 :2375 SRC/O.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
:2376
:2377 WR.PLUS.Q.XCHG:
C 019C, 010 :2377 SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 019D, 010 :2378 SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 019E, 010 :2379 SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 019F, 010 :2380 SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
:2381
:2382 WR.FROM.LS-1:
C 01A0, 14A :2383 SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01A1, 14A :2384 SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01A2, 14A :2385 SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01A3, 14A :2386 SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
```


	:2387	LS.FROM.WR.LS:
C 01A4, 349	:2388	SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01A5, 349	:2389	SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01A6, 349	:2390	SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01A7, 349	:2391	SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
	:2392	WR.PLUS.LS+1:
C 01A8, 348	:2393	SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01A9, 348	:2394	SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01AA, 348	:2395	SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01AB, 348	:2396	SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
	:2397	WR.FROM.LS:
C 01AC, 34A	:2398	SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01AD, 34A	:2399	SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01AE, 34A	:2400	SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01AF, 34A	:2401	SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
	:2402	
	:2403	WR.PLUS.LS:
C 01B0, 148	:2404	SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01B1, 148	:2405	SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01B2, 148	:2406	SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01B3, 148	:2407	SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
	:2408	WR.OR.LS:
C 01B4, 34B	:2409	SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01B5, 34B	:2410	SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01B6, 34B	:2411	SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01B7, 34B	:2412	SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
	:2413	WR.AND.LS:
C 01B8, 34C	:2414	SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 01B9, 34C	:2415	SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 01BA, 34C	:2416	SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 01BB, 34C	:2417	SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
	:2418	WR.XOR.LS:
C 01BC, 34E	:2419	SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01BD, 34E	:2420	SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01BE, 34E	:2421	SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01BF, 34E	:2422	SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
	:2423	
	:2424	Q.PLUS.LS:
C 01C0, 188	:2425	SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01C1, 188	:2426	SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01C2, 188	:2427	SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01C3, 188	:2428	SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
	:2429	LS.FROM.Q.LS:
C 01C4, 389	:2430	SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01C5, 389	:2431	SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01C6, 389	:2432	SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01C7, 389	:2433	SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
	:2434	Q.FROM.LS:
C 01C8, 38A	:2435	SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01C9, 38A	:2436	SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01CA, 38A	:2437	SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01CB, 38A	:2438	SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
	:2439	Q.OR.LS:
C 01CC, 38B	:2440	SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01CD, 38B	:2441	SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN

ZZ-ENKCA-1.0	:	ENKCA.MIC	CONTROL ROM CONTENT	M 5	Fiche 1	Frame M5	Sequence 64	Page 58
: ENKCA.MCR	:	MICRO2 1L(03)	3-MAR-82 09:34:33	ENKCA Micro-diagnostic	for DAP module	Rev 01.00		
: ENKCA.MIC	:	CONTROL ROM CONTENTS		VER 00.01				

C 01CE, 38B	:2442	SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01CF, 38B	:2443	SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
	:2444	
	:2445	Q.AND.LS:
C 01D0, 18C	:2446	SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
C 01D1, 18C	:2447	SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
C 01D2, 18C	:2448	SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
C 01D3, 18C	:2449	SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
	:2450	Q.XOR.LS:
C 01D4, 38E	:2451	SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01D5, 38E	:2452	SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01D6, 38E	:2453	SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01D7, 38E	:2454	SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
	:2455	MOV.Q.LS:
C 01D8, 28B	:2456	SRC/O.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01D9, 28B	:2457	SRC/O.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01DA, 28B	:2458	SRC/O.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01DB, 28B	:2459	SRC/O.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
	:2460	Q.NEG.LS:
C 01DC, 28A	:2461	SRC/O.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01DD, 28A	:2462	SRC/O.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01DE, 28A	:2463	SRC/O.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01DF, 28A	:2464	SRC/O.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
	:2465	
	:2466	WR.COM.LS:
C 01E0, 0CF	:2467	SRC/O.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
C 01E1, 0CF	:2468	SRC/O.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
C 01E2, 0CF	:2469	SRC/O.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
C 01E3, 0CF	:2470	SRC/O.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
	:2471	WR.NEG.LS:
C 01E4, 2CA	:2472	SRC/O.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01E5, 2CA	:2473	SRC/O.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01E6, 2CA	:2474	SRC/O.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01E7, 2CA	:2475	SRC/O.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
	:2476	Q.PLUS.WR.TO.LS:
C 01E8, 008	:2477	SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01E9, 008	:2478	SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01EA, 008	:2479	SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01EB, 008	:2480	SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
	:2481	Q.FROM.WR.TO.LS:
C 01EC, 20A	:2482	SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01ED, 20A	:2483	SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01EE, 20A	:2484	SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01EF, 20A	:2485	SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
	:2486	
	:2487	DEC.LS:
C 01F0, 1CA	:2488	SRC/D.O,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01F1, 1CA	:2489	SRC/D.O,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01F2, 1CA	:2490	SRC/D.O,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01F3, 1CA	:2491	SRC/D.O,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
	:2492	COM.LS:
C 01F4, 3CF	:2493	SRC/D.O,ALU/R.XNOR.S,DST/NOP,CIN/CIN
C 01F5, 3CF	:2494	SRC/D.O,ALU/R.XNOR.S,DST/NOP,CIN/CIN
C 01F6, 3CF	:2495	SRC/D.O,ALU/R.XNOR.S,DST/NOP,CIN/CIN
C 01F7, 3CF	:2496	SRC/D.O,ALU/R.XNOR.S,DST/NOP,CIN/CIN

	:2497	NEG.LS:	
C 01F8, 3C9	:2498		SRC/D.O,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01F9, 3C9	:2499		SRC/D.O,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01FA, 3C9	:2500		SRC/D.O,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01FB, 3C9	:2501		SRC/D.O,ALU/S.MINUS.R,DST/NOP,CIN/CIN
	:2502	INC.LS:	
C 01FC, 3C8	:2503		SRC/D.O,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01FD, 3C8	:2504		SRC/D.O,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01FE, 3C8	:2505		SRC/D.O,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01FF, 3C8	:2506		SRC/D.O,ALU/R.PLUS.S,DST/NOP,CIN/CIN
	:2507		
	:2508	.UCODE	


```

:2509 .PAGE "DIAGNOSTIC MACROS AND DEFINITIONS"
:2510
:2511 D.ADRS/=<15:9>,.VALIDITY=<D.VAL>,.DEFAULT=0
:2512
:2513
:2514 SAV.WR0=1 : STORAGE FOR WR0
:2515 SAV.WR1=2 : STORAGE FOR WR1
:2516 SAV.WR2=3 : STORAGE FOR WR2
:2517 SAV.WR3=4 : STORAGE FOR WR3
:2518 T5.SUB=5 : RESERVED FOR COMMON SUBROUTINES
:2519 T6.SUB=6 : RESERVED FOR COMMON SUBROUTINES
:2520 T7.SUB=7 : RESERVED FOR COMMON SUBROUTINES
:2521 TRANSFER.POINTER=7 : POINTER FOR TRANSFER ROUTINE
:2522 MEMORY.SIZE=7 : STORAGE FOR MEMORY SIZE IN 256KB BLOCKS AFTER
:2523 : SIZING
:2524 FPA.FOUND=7 : STORAGE FOR RESULT OF SIZING FOR FPA PRESENT
:2525 UBE.FOUND=7 : STORAGE FOR RESULT OF SIZING FOR UBE PRESENT
:2526 T8=8 : TEMP LOCATION 8
:2527 T9=9 : TEMP LOCATION 9
:2528 T10=0A : TEMP LOCATION 10
:2529 T11=0B : TEMP LOCATION 11
:2530 T12=0C : TEMP LOCATION 12
:2531 T13=0D : TEMP LOCATION 13
:2532 T14=0E : TEMP LOCATION 14
:2533 T15=0F : TEMP LOCATION 15
:2534 PC=10 : MACRO PROGRAM COUNTER
:2535
:2536 WR.BACKUP=11 : BACKUP WR FOR MOV MEM.DATA TO WR[]
:2537 MM.LASTADDR+1=12 : STORAGE OF LAST ADDRESS IN MAIN MEMORY PLUS
:2538 : 1 (FIRST NON-EXISTANT MEMORY ADDRESS)
:2539 #FF=13 : MASK
:2540 #FFFF=14 : MASK
:2541 #FF000000=15 : MASK
:2542 #FFFFFF00=16 : MASK
:2543 CSR0.MASK=16 : MASK
:2544 CSR1.MASK=17 : MASK (01003FFF)
:2545 CSR2.MASK=18 : MASK (7FFE3FFF)
:2546 #FE7FFFFFFF=19 : MASK
:2547 UBS.SET=1A : CONSTANT (7FF80000) USED BY UBS ADDRESS TEST
:2548 UBS.DATA=1B : MASK (7FFF8000) USED AS ERROR MASK FOR UBS
:2549 : DATA TEST
:2550
:2551 ERR.CON.NA=1E : CONSTANT OF 800500(H) BITS 23,10,8 SET FOR
:2552 : LOADING CONTROL AND STATUS REG IN INITIAL
:2553 : TESTS
:2554 ERR.CON=1F : CONSTANT OF 500(H) BITS 10 & 8 SET FOR
:2555 : LOADING CONTROL AND STATUS REG IN INITIAL
:2556 : TESTS
:2557
:2558 #1=20 : PATTERN 00000001 (HEX)
:2559 #2=21 : PATTERN 00000002
:2560 #4=22 : PATTERN 00000004
:2561 #8=23 : PATTERN 00000008
:2562 #10=24 : PATTERN 00000010
:2563 #20=25 : PATTERN 00000020

```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

DIAGNOSTIC MACROS A 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC MACROS AND DEFINITIONS

Fiche 1 Frame C6
ENKCA Micro-diagnostic for DAP module
Sequence 67
Rev 01.00
Page 61

:2564	#40=26	: PATTERN 00000040
:2565	#80=27	: PATTERN 00000080
:2566	#100=28	: PATTERN 00000100
:2567	#200=29	: PATTERN 00000200
:2568	#400=2A	: PATTERN 00000400
:2569	#800=2B	: PATTERN 00000800
:2570	#1000=2C	: PATTERN 00001000
:2571	#2000=2D	: PATTERN 00002000
:2572	#4000=2E	: PATTERN 00004000
:2573	#8000=2F	: PATTERN 00008000
:2574	#10000=30	: PATTERN 00010000
:2575	#20000=31	: PATTERN 00020000
:2576	#40000=32	: PATTERN 00040000
:2577	#80000=33	: PATTERN 00080000
:2578	#100000=34	: PATTERN 00100000
:2579	#200000=35	: PATTERN 00200000
:2580	#400000=36	: PATTERN 00400000
:2581	#800000=37	: PATTERN 00800000
:2582	#1000000=38	: PATTERN 01000000
:2583	#2000000=39	: PATTERN 02000000
:2584	#4000000=3A	: PATTERN 04000000
:2585	#8000000=3B	: PATTERN 08000000
:2586	#10000000=3C	: PATTERN 10000000
:2587	#20000000=3D	: PATTERN 20000000
:2588	#40000000=3E	: PATTERN 40000000
:2589	#80000000=3F	: PATTERN 80000000
:2590		
:2591	BIT0=20	: PATTERN 00000001
:2592	BIT1=21	: PATTERN 00000002
:2593	BIT2=22	: PATTERN 00000004
:2594	BIT3=23	: PATTERN 00000008
:2595	BIT4=24	: PATTERN 00000010
:2596	BIT5=25	: PATTERN 00000020
:2597	BIT6=26	: PATTERN 00000040
:2598	BIT7=27	: PATTERN 00000080
:2599	BIT8=28	: PATTERN 00000100
:2600	BIT9=29	: PATTERN 00000200
:2601	BIT10=2A	: PATTERN 00000400
:2602	BIT11=2B	: PATTERN 00000800
:2603	BIT12=2C	: PATTERN 00001000
:2604	BIT13=2D	: PATTERN 00002000
:2605	BIT14=2E	: PATTERN 00004000
:2606	BIT15=2F	: PATTERN 00008000
:2607	BIT16=30	: PATTERN 00010000
:2608	BIT17=31	: PATTERN 00020000
:2609	BIT18=32	: PATTERN 00040000
:2610	BIT19=33	: PATTERN 00080000
:2611	BIT20=34	: PATTERN 00100000
:2612	BIT21=35	: PATTERN 00200000
:2613	BIT22=36	: PATTERN 00400000
:2614	BIT23=37	: PATTERN 00800000
:2615	BIT24=38	: PATTERN 01000000
:2616	BIT25=39	: PATTERN 02000000
:2617	BIT26=3A	: PATTERN 04000000
:2618	BIT27=3B	: PATTERN 08000000

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

DIAGNOSTIC MACROS A 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC MACROS AND DEFINITIONS

ENKCA Micro-diagnostic for DAP module

Sequence 68
Rev 01.00

Page 62

```
:2619          BIT28=3C          : PATTERN 10000000
:2620          BIT29=3D          : PATTERN 20000000
:2621          BIT30=3E          : PATTERN 40000000
:2622          BIT31=3F          : PATTERN 80000000
:2623
:2624
:2625      :CSR ADDRESS MNEMONICS
:2626
:2627          CSR0=4E          : ALL BITS CLEAR TO ACCESS CSR 0
:2628          CSR1=22          : BIT 2 SET TO ACCESS CSR 1
:2629          CSR2=23          : BIT 3 SET TO ACCESS CSR 2
:2630
:2631      :CONTROL AND STATUS WORD BITS
:2632
:2633          PRINT.UBE=26      : PRINT IF UBE PRESENT OR NOT PRESENT (BIT 6)
:2634                                (INDICATOR IN LS 7)
:2635          PRINT.R80=27      : PRINT IF R80 PRESENT OR NOT PRESENT AND ON
:2636                                WHICH DRIVE (BIT 7). INDICATOR IN LS 7,
:2637                                DRIVE NUMBER IN LS 6.
:2638          ERROR=28          : TEST ERROR INDICATION (BIT 8)
:2639          SET.PA.ERR=29      : TELLS CONSOLE TO CREATE A PARITY ERROR AT WCS
:2640                                ADDRESS POINTED TO BY LS 7 (BIT 9)
:2641          CONSOLE.LOE=2A      : INDICATES CONSOLE DOES ERROR LOOPING (BIT 10)
:2642                                (FOR INITIAL TESTS)
:2643          PRINT.MEM.SIZE=2B    : PRINT MEMORY SIZE IN 256K BLOCKS (BIT 11)
:2644                                (SIZE IN LS 7)
:2645          PRINT.FPA=2C      : PRINT IF FPA PRESENT OR NOT PRESENT (BIT 12)
:2646                                (INDICATOR IN LS 7)
:2647          XFER.DATA=2D      : INDICATES A DATA TRANSFER TO LS OR MM (BIT 13)
:2648          EOS=2E          : END OF SECTION (BIT 14)
:2649          BEGIN.TEST=2F      : BEGINNING OF TEST (BIT 15)
:2650          INTERRUPT.EN=30      : ENABLE INTERRUPT OR SIGNAL SPECIFIED BY BITS
:2651                                17 THRU 22 AND 28 THRU 30 (BIT 16)
:2652
:2653          CONSOLE.HALT=31      : CONSOLE HALT INTERRUPT (BIT 17)
:2654          PWR.FAIL=32          : PWR FAIL INT (BIT 18)
:2655          INTERVAL.TIM=33      : INTRVL TIM INT (BIT 19)
:2656          CONSOLE.ATTN=34      : CONSOLE ATTN INTERRUPT (BIT 20)
:2657          CONSOLE.ACK=35      : CONSOLE ACK INTERRUPT (BIT 21)
:2658          DISABLE.MEM.REF=36    : DISABLE MEMORY REFERENCES (BIT 22)
:2659          CINIT=3C          : UNIBUS INIT SIGNAL TO THE MCT (BIT 28)
:2660          UBS.DCLO=3D          : UNIBUS DC LO INDICATION TO THE MCT (BIT 29)
:2661          UBS.BBSY=3E          : UNIBUS BUS BUSY INDICATOR TO MCT (BIT 30)
:2662
:2663          NA.EXP.REC=37      : PRINT N/A UNDER EXPECTED AND RECEIVED (BIT 23)
:2664          OTHER.DATA=38      : PRINT A VALUE UNDER OTHER DATA (BIT 24)
:2665          CONSOLE.LOOP=39      : CONSOLE PERFORMS A LOOP ACCORDING TO LOOP
:2666                                COUNT IN LS (BIT 25)
:2667          PRINT.CRD=3A          : PRINT NUMBER OF SINGLE BIT ERRORS (BIT 26)
:2668          CLR.PA.ERR=3B      : TELLS CONSOLE TO CLEAR PARITY ERROR AT WCS
:2669                                ADDRESS POINTED TO BY LS 7 (BIT 27)
:2670          PRINT.IDC=3F      : PRINT IF IDC PRESENT OR NOT PRESENT (BIT 31)
:2671                                (INDICATOR IN LS 7)
:2672
:2673      :MODULE CODES
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

DIAGNOSTIC MACROS A 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC MACROS AND DEFINITIONS

E 6

ENKCA Micro-diagnostic for DAP module

Fiche 1 Frame E6

Sequence 69
Rev 01.00

Page 63

```
:2674
:2675      WCS=20      : MODULE CODE FOR WCS BOARD (BIT 0 SET)
:2676      CPU=21      : MODULE CODE FOR CPU BOARD (BIT 1 SET)
:2677      MCT=22      : MODULE CODE FOR MCT BOARD (BIT 2 SET)
:2678      FPA=23      : MODULE CODE FOR FPA BOARD (BIT 3 SET)
:2679      ARRAY=24    : MODULE CODE FOR ARRAY BOARD (BIT 4 SET)
:2680      IDC=25      : MODULE CODE FOR IDC BOARD (BIT 5 SET)
:2681
:2682      ;CSR 1 ERROR BITS
:2683
:2684      VALID.ERR=2E  : VALID NOT SET IF 1 (BIT 14)
:2685      TB.PAR.ERR=2F : TB PARITY ERROR (BIT 15)
:2686      NXM=30       : NON-EXISTANT MEMORY ERROR (BIT 16)
:2687      UB.BSY=31    : UNIBUS BUSY (BIT 17)
:2688      ADP.REG=32    : UNIBUS ADAPTER REGISTER SELECT (BIT 18)
:2689      WR.ACROSS.PG=33 : WRITE ACROSS PAGE BOUNDARY ERROR (BIT 19)
:2690      OP.ERR=34     : OPERATION ERROR (BIT 20)
:2691      TB.MISS=35    : TB MISS (VIRTUAL ADDRESS TRANSLATION NOT IN
:2692                   : BUFFER) (BIT 21)
:2693      ACCESS.REFUSED=36 : PROTECTION ERROR ON ACCESS (BIT 22)
:2694      MODIFY.REFUSED=37 : PROTECTION ERROR ON WRITE (BIT 23)
:2695
:2696      CRD=3E         : SINGLE BIT ERROR CORRECTED (BIT 30)
:2697      RDS=3F         : UNCORRECTABLE DATA ERROR (BIT 31)
:2698
:2699
:2700      ;CSR 1 CONTROL BITS
:2701
:2702      ECC.DIS=39     : CSR1 ECC DISABLE BIT (BIT 25)
:2703      DIAG.CHK=3A   : CSR1 DIAG CHK BIT (BIT 26)
:2704      MME=3B        : CSR1 MEMORY MANAGMENT ENABLE BIT (BIT 27)
:2705      INH.CRD=3C    : CSR1 INHIBIT REPORT CRD BIT (BIT 28)
:2706      TB.PAR.DIAG=3D : CSR1 TB PAR DIAG BIT (FORCE TB PARITY ERR)
:2707                   : (BIT 29)
:2708
:2709
:2710      ;UBE CONTROL REGISTER BITS
:2711
:2712      ;CONTROL REG 1
:2713      GO=20         : START UBE (BIT 0)
:2714      BR4=21        : ISSUE BUS REQUEST 4 (BIT 1)
:2715      BR5=22        : ISSUE BUS REQUEST 5 (BIT 2)
:2716      BR6=23        : ISSUE BUS REQUEST 6 (BIT 3)
:2717      BR7=24        : ISSUE BUS REQUEST 7 (BIT 4)
:2718      NPR=25        : ISSUE NPR (BIT 5)
:2719      INT.ODNE=26    : INTERRUPT ON DONE (WHEN CCOVF) (BIT 6)
:2720      RDY=27         : READY BIT, SET WHEN READY TO BEGIN FUNC (BIT 7)
:2721      CO0=28         : C0 BIT (BIT 8)
:2722      CO1=29         : C1 BIT (BIT 9)
:2723      FUNA=2A        : FUNCTION BIT A (BIT 10)
:2724      FUNB=2B        : FUNCTION BIT B (BIT 11)
:2725      CC+DAT=2C      : DATA FROM CYCLE COUNT IF SET, DATA REG IF CLEAR
:2726      INH.DATIP.ROL=2D : INHIBIT ROL ON DATIP (BIT 13)
:2727      DATOB.ON.DATIP=2E : DO DATOB AFTER DATIP CYCLE (BIT 14)
:2728      ERR=2F         : EXERCISOR OR UNIBUS ERROR (BIT 15)
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

DIAGNOSTIC MACROS A 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC MACROS AND DEFINITIONS

F 6

ENKCA Micro-diagnostic for DAP module

Fiche 1 Frame F6

Sequence 70
Rev 01.00

Page 64

```
:2729
:2730      :CONTROL REG 2
:2731      EXT.MEM0=20      : EXTENDED MEMORY BIT 0 (BIT 0)
:2732      EXT.MEM1=21      : EXTENDED MEMORY BIT 1 (BIT 1)
:2733      INH.INC.ADDR&CC=22 : INHIBIT INC ON CYCLE COUNT AND ADDR REG (BIT 2)
:2734      INH.SACK=23      : INHIBIT ASSERTING BUS SACK ON BUS GRANT (BIT 3)
:2735      PWR.DWN.SEQ=24    : ASSERT ACLC (BIT 4)
:2736      WNG.GNT.BCK=25    : NO BUS GRANT RECV TO HIGEST REQUEST (BIT 5)
:2737      MAX.LATE.ERR=26   : NPR AND BR LATENCY TIME EXCEEDED (BIT 6)
:2738      NO.SACK.ERR=27    : NO TIMEOUT AT CPU WHEN NO SACK (BIT 7)
:2739      NO.SSYN.ERR=28    : NO SSYN RECEIVED AFTER ASSERTING MSYN (BIT 8)
:2740      WRG.A.LINES=29    : ADDRESS ERR ON SENT & RECEIVED ADDRESS (BIT 9)
:2741      NO.GNT+NOT.ONE.GNT=2A : NO GRANT OR MORE THAN 1 GRANT AFTER REQ. (BIT 10)
:2742      INTR.SSYN.ERR=2B  : NO SSYN RECEIVED AFTER BUS INTR (BIT 11)
:2743      ENB.PB=2C         : ENABLE PARITY BIT B (BIT 12)
:2744      CCOVF=2D          : CYCLE COUNT OVERFLOW (BIT 13)
:2745      TM.DLY=2E         : REQUEST BUS EVERY 10 USEC (BIT 14)
:2746
:2747      :BG6 MNEMONIC
:2748      BG6=23            : BIT 3 SET FOR BG 6 ADDRESS
:2749
:2750      :ERROR AND CONTROL INFORMATION
:2751
:2752      CONTROL.STATUS=40 : CONTROL AND STATUS WORD
:2753      ERROR.NUMBER=41   : ERROR NUMBER OF CURRENT TEST
:2754      EXPECTED.DATA=42 : EXPECTED RESULT OF CURRENT TEST
:2755      RECEIVED.DATA=43 : ACTUAL RESULT OF CURRENT TEST
:2756      ADDRESS.DATA=44  : OTHER PERTINENT DATA
:2757      ERROR.MASK=45     : BITS TO CHECK IN RESULT
:2758      MODULE.NUM=46    : STORAGE OF MODULE NUMBER (CODED)
:2759      LOOP.CNT=47      : STORAGE OF LOOP COUNT FOR CONSOLE LOOP
:2760
:2761      ERROR.CONTROL=48  : CONTROL
:2762      LOE=20            : STORAGE FOR ERROR CONTROL (LOOP ON ERROR ETC.)
:2763      NER=21            : LOOP ON ERROR IF SET (BIT0)
:2764      BELL=22           : NO ERROR REPORT IF SET (BIT1)
:2765      HOE=23           : BELL ON ERROR IF SET (BIT2)
:2766      SER=24           : HALT ON ERROR IF SET (BIT3)
:2767      APT.PRESENT=25    : ENABLE ERROR REPORT ON CRD ERRORS IF SET
:2768      LOOP.COMMAND=26  : APT PRESENT IF SET (BIT5)
:2769
:2770      APT.PARAM=49      : APT RUN TIME PARAMETER REGS (MEM SIZE, HARD-
:2771      :WARE CONFIG, SOFTWARE CONFIG IN BYTES 0, 1
:2772      :AND 2 RESPECTIVELY. BYTE 3 FOR FUTURE USE)
:2773      APT.RESERVED=4A   : RESERVED FOR FUTURE APT USE
:2774
:2775      :MISCELLANEOUS CONSTANTS AND STORAGE
:2776
:2777      #AAAAAAAA=4C       : CONSTANT
:2778      ALTER.ODD=4C      : CONSTANT
:2779      #55555555=4D      : CONSTANT
:2780      ALTER.EVEN=4D     : CONSTANT
:2781      #0=4E             : CONSTANT
:2782      ZERO=4E           : CONSTANT
:2783      #FFFFFFFF=4F       : CONSTANT
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

DIAGNOSTIC MACROS A 3-^{G 6}MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC MACROS AND DEFINITIONS

Fiche 1 Frame G6

Sequence 71
Rev 01.00

Page 65

```
:2784      ONES=4F      : CONSTANT
:2785      PREVIOUS.ERROR=50      : ERROR NUMBER OF LAST ERROR
:2786
:2787      DATA.1=51      : STORAGE FOR FPA DATA
:2788      DATA.2=52      : STORAGE FOR FPA DATA
:2789      DATA.3=53      : STORAGE FOR FPA DATA
:2790      FIRST.FLT=54      : STORAGE FOR FPA DATA
:2791      SEC.FLT=55      : STORAGE FOR FPA DATA
:2792      THIRD.FLT=56      : STORAGE FOR FPA DATA
:2793      T57=57      : TEMP LOCATION 57
:2794      T58=58      : TEMP LOCATION 58
:2795      T59=59      : TEMP LOCATION 59
:2796
:2797      BEDB=5A      :UBE (BUS EXERCISOR) DATA BUF REG
:2798      BECC=5B      :UBE (BUS EXERCISOR) CYCLE COUNT REG
:2799      BEBA=5C      :UBE (BUS EXERCISOR) ADDR REG
:2800      BECR1=5D      :UBE (BUS EXERCISOR) CONTROL REG 1
:2801      CLEAR.ERR.ADDR=5E      :UBE CLEAR ERROR REG ADDRESS
:2802      BECR2=5F      :UBE (BUS EXERCISOR) CONTROL REG 2
:2803
:2804      #3(H)=60      : CONSTANT
:2805      #12(H)=61      : CONSTANT
:2806      #33333333=62      : CONSTANT
:2807      #0F0F0F0F=63      : CONSTANT
:2808      #00FF00FF=64      : CONSTANT
:2809      #162(H)=65      : CONSTANT FOR LS TRANSFER POINTER
:2810      BR7.IDENT=66      : BR7 IDENTIFIER (1010(B) IN BITS 5-2)
:2811      BG7=67      : BG7 ADDRESS (BITS 2 AND 3 SET)
:2812
:2813      :TEMPS FOR CPU TESTS
:2814      L30=30      :LS 30 TEMP (RESTORED TO 10000 AFTER USE)
:2815      L31=31      :LS 31 TEMP (RESTORE TO 20000 AFTER USE)
:2816      L53=53      :LS 53 TEMP
:2817      L73=73      :LS 73 TEMP
:2818
:2819      :REGISTER ADDRESSES
:2820
:2821      CONTROL.OS=78      : HARDWARE INDEX TO CONTROL WORDS (LS 40-4F)
:2822      SHIFT.OS(3-0)=79      : 16 LOCATION HARDWARE INDEX TO SHIFT PATTERN
:2823      : (LS 20-2F)
:2824      SHIFT.OS(4-0)=7B      : 32 LOCATION HARDWARE INDEX TO SHIFT PATTERN
:2825      : (LS 20-3F)
:2826      OS=7C      : OS REGISTER
:2827      DEC.CON=7D      : DECIMAL CARRIES
:2828      SIZE=7E      : SIZE REGISTER
:2829      MDT= 7E      : MEMORY REFERENCE DATA SIZE
:2830      INTERRUPT.VEC=7E      : HARDWARE INTERRUPT VECTOR
:2831      :
:2832      : THIS WORD IS THE HARDWARE CONDITION CODES. THE CC'S CAN BE READ
:2833      : OR ARE WRITTEN HERE. NO OTHER BITS IN THE PSL ARE EFFECTED.
:2834      :
:2835      PSL.CC=7F      : PSL CONDITION CODES
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

DIAGNOSTIC MACROS A 3-H 6
MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC MACROS AND DEFINITIONS

ENKCA Micro-diagnostic for DAP module

Sequence 72
Rev 01.00

Page 66

```
:2836 .PAGE
:2837 XD.ADRS/=<16:9>,.VALIDITY=<XD.VAL>
:2838
:2839
:2840 SAV.WR0=1      : STORAGE FOR WR0
:2841 SAV.WR1=2      : STORAGE FOR WR1
:2842 SAV.WR2=3      : STORAGE FOR WR2
:2843 SAV.WR3=4      : STORAGE FOR WR3
:2844 T5.SUB=5        : RESERVED FOR COMMON SUBROUTINES
:2845 T6.SUB=6        : RESERVED FOR COMMON SUBROUTINES
:2846 T7.SUB=7        : RESERVED FOR COMMON SUBROUTINES
:2847 TRANSFER.POINTER=7 : POINTER FOR TRANSFER ROUTINE
:2848 MEMORY.SIZE=7    : STORAGE FOR MEMORY SIZE IN 256KB BLOCKS AFTER
:2849                  : SIZING
:2850 FPA.FOUND=7      : STORAGE FOR RESULT OF SIZING FOR FPA PRESENT
:2851 UBE.FOUND=7      : STORAGE FOR RESULT OF SIZING FOR UBE PRESENT
:2852 T8=8            : TEMP LOCATION 8
:2853 T9=9            : TEMP LOCATION 9
:2854 T10=0A          : TEMP LOCATION 10
:2855 T11=0B          : TEMP LOCATION 11
:2856 T12=0C          : TEMP LOCATION 12
:2857 T13=0D          : TEMP LOCATION 13
:2858 T14=0E          : TEMP LOCATION 14
:2859 T15=0F          : TEMP LOCATION 15
:2860 PC=10          : MACRO PROGRAM COUNTER
:2861
:2862 WR.BACKUP=11     : BACKUP WR FOR MOV MEM.DATA TO WR[]
:2863 MM.LASTADDR+1=12 : STORAGE OF LAST ADDRESS IN MAIN MEMORY PLUS
:2864                  : 1 (FIRST NON-EXISTANT MEMORY ADDRESS)
:2865 #FF=13           : MASK
:2866 #FFFF=14         : MASK
:2867 #FF000000=15     : MASK
:2868 #FFFFFF00=16     : MASK
:2869 CSR0.MASK=16     : MASK
:2870 CSR1.MASK=17     : MASK (01003FFF)
:2871 CSR2.MASK=18     : MASK (FFFE3FFF)
:2872 #FE7FFFFFFF=19  : MASK
:2873 UBS.SET=1A       : CONSTANT (7FF80000) USED BY UBS ADDRESS TEST
:2874 UBS.DATA=1B     : MASK (7FFF8000) USED AS ERROR MASK FOR UBS
:2875                  : DATA TEST
:2876
:2877 ERR.CON.NA=1E    : CONSTANT OF 800500(H) BITS 23,10,8 SET FOR
:2878                  : LOADING CONTROL AND STATUS REG IN INITIAL
:2879                  : TESTS
:2880 ERR.CON=1F        : CONSTANT OF 500(H) BITS 10 & 8 SET FOR
:2881                  : LOADING CONTROL AND STATUS REG IN INITIAL
:2882                  : TESTS
:2883
:2884 #1=20             : PATTERN 00000001 (HEX)
:2885 #2=21             : PATTERN 00000002
:2886 #4=22             : PATTERN 00000004
:2887 #8=23             : PATTERN 00000008
:2888 #10=24            : PATTERN 00000010
:2889 #20=25            : PATTERN 00000020
:2890 #40=26            : PATTERN 00000040
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

DIAGNOSTIC MACROS A 3-¹MAR-82⁶
MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC MACROS AND DEFINITIONS

Fiche 1 Frame I6
ENKCA Micro-diagnostic for DAP module
Sequence 73
Rev 01.00
Page 67

:2891	#80=27	: PATTERN 0C000080
:2892	#100=28	: PATTERN 00000100
:2893	#200=29	: PATTERN 00000200
:2894	#400=2A	: PATTERN 00000400
:2895	#800=2B	: PATTERN 00000800
:2896	#1000=2C	: PATTERN 00001000
:2897	#2000=2D	: PATTERN 00002000
:2898	#4000=2E	: PATTERN 00004000
:2899	#8000=2F	: PATTERN 00008000
:2900	#10000=30	: PATTERN 00010000
:2901	#20000=31	: PATTERN 00020000
:2902	#40000=32	: PATTERN 00040000
:2903	#80000=33	: PATTERN 00080000
:2904	#100000=34	: PATTERN 00100000
:2905	#200000=35	: PATTERN 00200000
:2906	#400000=36	: PATTERN 00400000
:2907	#800000=37	: PATTERN 00800000
:2908	#1000000=38	: PATTERN 01000000
:2909	#2000000=39	: PATTERN 02000000
:2910	#4000000=3A	: PATTERN 04000000
:2911	#8000000=3B	: PATTERN 08000000
:2912	#10000000=3C	: PATTERN 10000000
:2913	#20000000=3D	: PATTERN 20000000
:2914	#40000000=3E	: PATTERN 40000000
:2915	#80000000=3F	: PATTERN 80000000
:2916		
:2917	BIT0=20	: PATTERN 00000001
:2918	BIT1=21	: PATTERN 00000002
:2919	BIT2=22	: PATTERN 00000004
:2920	BIT3=23	: PATTERN 00000008
:2921	BIT4=24	: PATTERN 00000010
:2922	BIT5=25	: PATTERN 00000020
:2923	BIT6=26	: PATTERN 00000040
:2924	BIT7=27	: PATTERN 00000080
:2925	BIT8=28	: PATTERN 00000100
:2926	BIT9=29	: PATTERN 00000200
:2927	BIT10=2A	: PATTERN 00000400
:2928	BIT11=2B	: PATTERN 00000800
:2929	BIT12=2C	: PATTERN 00001000
:2930	BIT13=2D	: PATTERN 00002000
:2931	BIT14=2E	: PATTERN 00004000
:2932	BIT15=2F	: PATTERN 00008000
:2933	BIT16=30	: PATTERN 00010000
:2934	BIT17=31	: PATTERN 00020000
:2935	BIT18=32	: PATTERN 00040000
:2936	BIT19=33	: PATTERN 00080000
:2937	BIT20=34	: PATTERN 00100000
:2938	BIT21=35	: PATTERN 00200000
:2939	BIT22=36	: PATTERN 00400000
:2940	BIT23=37	: PATTERN 00800000
:2941	BIT24=38	: PATTERN 01000000
:2942	BIT25=39	: PATTERN 02000000
:2943	BIT26=3A	: PATTERN 04000000
:2944	BIT27=3B	: PATTERN 08000000
:2945	BIT28=3C	: PATTERN 10000000


```

:3001 :MODULE CODES
:3002
:3003     WCS=20      : MODULE CODE FOR WCS BOARD (BIT 0 SET)
:3004     CPU=21     : MODULE CODE FOR CPU BOARD (BIT 1 SET)
:3005     MCT=22     : MODULE CODE FOR MCT BOARD (BIT 2 SET)
:3006     FPA=23     : MODULE CODE FOR FPA BOARD (BIT 3 SET)
:3007     ARRAY=24   : MODULE CODE FOR ARRAY BOARD (BIT 4 SET)
:3008     IDC=25     : MODULE CODE FOR IDC BOARD (BIT 5 SET)
:3009
:3010 :CSR 1 ERROR BITS
:3011
:3012     VALID.ERR=2E : VALID NOT SET IF 1 (BIT 14)
:3013     TB.PAR.ERR=2F : TB PARITY ERROR (BIT 15)
:3014     NXM=30       : NON-EXISTANT MEMORY ERROR (BIT 16)
:3015     UB.BSY=31    : UNIBUS BUSY (BIT 17)
:3016     ADP.REG=32   : UNIBUS ADAPTER REGISTER SELECT (BIT 18)
:3017     WR.ACROSS.PG=33 : WRITE ACROSS PAGE BOUNDARY ERROR (BIT 19)
:3018     OP.ERR=34    : OPERATION ERROR (BIT 20)
:3019     TB.MISS=35   : TB MISS (VIRTUAL ADDRESS TRANSLATION NOT IN
:3020                   : BUFFER) (BIT 21)
:3021     ACCESS.REFUSED=36 : PROTECTION ERROR ON ACCESS (BIT 22)
:3022     MODIFY.REFUSED=37 : PROTECTION ERROR ON WRITE (BIT 23)
:3023
:3024     CRD=3E       : SINGLE BIT ERROR CORRECTED (BIT 30)
:3025     RDS=3F      : UNCORRECTABLE DATA ERROR (BIT 31)
:3026
:3027 :CSR 1 CONTROL BITS
:3028
:3029
:3030     ECC.DIS=39    : CSR1 ECC DISABLE BIT (BIT 25)
:3031     DIAG.CHK=3A  : CSR1 DIAG CHK BIT (BIT 26)
:3032     MME=3B       : CSR1 MEMORY MANAGMENT ENABLE BIT (BIT 27)
:3033     INH.CRD=3C   : CSR1 INHIBIT REPORT CRD BIT (BIT 28)
:3034     TB.PAR.DIAG=3D : CSR1 TB PAR DIAG BIT (FORCE TB PARITY ERR)
:3035                   : (BIT 29)
:3036
:3037 :UBE CONTROL REGISTER BITS
:3038
:3039
:3040 :CONTROL REG 1
:3041     GO=20        : START UBE (BIT 0)
:3042     BR4=21       : ISSUE BUS REQUEST 4 (BIT 1)
:3043     BR5=22       : ISSUE BUS REQUEST 5 (BIT 2)
:3044     BR6=23       : ISSUE BUS REQUEST 6 (BIT 3)
:3045     BR7=24       : ISSUE BUS REQUEST 7 (BIT 4)
:3046     NPR=25       : ISSUE NPR (BIT 5)
:3047     INT.ODNE=26  : INTERRUPT ON DONE (WHEN CCOVF) (BIT 6)
:3048     RDY=27       : READY BIT, SET WHEN READY TO BEGIN FUNC (BIT 7)
:3049     C00=28       : C0 BIT (BIT 8)
:3050     C01=29       : C1 BIT (BIT 9)
:3051     FUNA=2A      : FUNCTION BIT A (BIT 10)
:3052     FUNB=2B      : FUNCTION BIT B (BIT 11)
:3053     CC+DAT=2C     : DATA FROM CYCLE COUNT IF SET, DATA REG IF CLEAR
:3054     INH.DATIP.ROL=2D : INHIBIT ROL ON DATIP (BIT 13)
:3055     DATOB.ON.DATIP=2E : DO DATOB AFTER DATIP CYCLE (BIT 14)
    
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC MACROS AND DEFINITIONS

DIAGNOSTIC MACROS A 3-MAR-82

Fiche 1 Frame L6

Sequence 76
Rev 01.00

Page 70

```
:3056          ERR=2F          : EXERCISOR OR UNIBUS ERROR (BIT 15)
:3057
:3058      :CONTROL REG 2
:3059          EXT.MEM0=20      : EXTENDED MEMORY BIT 0 (BIT 0)
:3060          EXT.MEM1=21      : EXTENDED MEMORY BIT 1 (BIT 1)
:3061          INH.INC.ADDR&CC=22 : INHIBIT INC ON CYCLE COUNT AND ADDR REG (BIT 2)
:3062          INH.SACK=23      : INHIBIT ASSERTING BUS SACK ON BUS GRANT (BIT 3)
:3063          PWR.DWN.SEQ=24    : ASSERT ACLO (BIT 4)
:3064          WNG.GNT.BCK=25    : NO BUS GRANT RECV TO HIGEST REQUEST (BIT 5)
:3065          MAX.LATE.ERR=26   : NPR AND BR LATENCY TIME EXCEEDED (BIT 6)
:3066          NO.SACK.ERR=27   : NO TIMEOUT AT CPU WHEN NO SACK (BIT 7)
:3067          NO.SSYN.ERR=28   : NO SSYN RECEIVED AFTER ASSERTING MSYN (BIT 8)
:3068          WRG.A.LINES=29   : ADDRESS ERR ON SENT & RECEIVED ADDRESS (BIT 9)
:3069          NO.GNT+NOT.ONE.GNT=2A : NO GRANT OR MORE THAN 1 GRANT AFTER REQ. (BIT 10)
:3070          INTR.SSYN.ERR=28  : NO SSYN RECEIVED AFTER BUS INTR (BIT 11)
:3071          ENB.PB=2C        : ENABLE PARITY BIT B (BIT 12)
:3072          CCOVF=2D        : CYCLE COUNT OVERFLOW (BIT 13)
:3073          TM.DLY=2E        : REQUEST BUS EVERY 10 USEC (BIT 14)
:3074
:3075      :BG6 MNEMONIC
:3076          BG6=23          : BIT 3 SET FOR BG 6 ADDRESS
:3077
:3078      :ERROR AND CONTROL INFORMATION
:3079
:3080          CONTROL.STATUS=40  : CONTROL AND STATUS WORD
:3081          ERROR.NUMBER=41    : ERROR NUMBER OF CURRENT TEST
:3082          EXPECTED.DATA=42  : EXPECTED RESULT OF CURRENT TEST
:3083          RECEIVED.DATA=43   : ACTUAL RESULT OF CURRENT TEST
:3084          ADDRESS.DATA=44    : OTHER PERTINENT DATA
:3085          ERROR.MASK=45     : BITS TO CHECK IN RESULT
:3086          MODULE.NUM=46     : STORAGE OF MODULE NUMBER (CODED)
:3087          LOOP.CNT=47       : STORAGE OF LOOP COUNT FOR CONSOLE LOOP
:3088
:3089          ERROR.CONTROL=48   : CONTROL
:3090          LOE=20            : STORAGE FOR ERROR CONTROL (LOOP ON ERROR ETC.)
:3091          NER=21            : LOOP ON ERROR IF SET (BIT0)
:3092          BELL=22           : NO ERROR REPORT IF SET (BIT1)
:3093          HOE=23           : BELL ON ERROR IF SET (BIT2)
:3094          SER=24           : HALT ON ERROR IF SET (BIT3)
:3095          APT.PRESENT=25    : ENABLE ERROR REPORT ON CRD ERRORS IF SET
:3096          LOOP.COMMAND=26  : APT PRESENT IF SET (BIT5)
:3097
:3098          APT.PARAM=49       : LOOP COMMAND TYPED IF SET (BIT 6)
:3099
:3100
:3101          APT.RESERVED=4A    : APT RUN TIME PARAMETER REGS (MEM SIZE, HARD-
:3102
:3103      :MISCELLANEOUS CONSTANTS AND STORAGE
:3104
:3105          #AAAAAAA=4C        : CONSTANT
:3106          ALTER.ODD=4C       : CONSTANT
:3107          #55555555=4D       : CONSTANT
:3108          ALTER.EVEN=4D      : CONSTANT
:3109          #0=4E             : CONSTANT
:3110          ZERO=4E           : CONSTANT
```

```

:3111          #FFFFFFF=4F          : CONSTANT
:3112          ONES=4F              : CONSTANT
:3113          PREVIOUS.ERROR=50    : ERROR NUMBER OF LAST ERROR
:3114
:3115          DATA.1=51           : STORAGE FOR FPA DATA
:3116          DATA.2=52           : STORAGE FOR FPA DATA
:3117          DATA.3=53           : STORAGE FOR FPA DATA
:3118          FIRST.FLT=54         : STORAGE FOR FPA DATA
:3119          SEC.FLT=55           : STORAGE FOR FPA DATA
:3120          THIRD.FLT=56         : STORAGE FOR FPA DATA
:3121          T57=57               : TEMP LOCATION 57
:3122          T58=58               : TEMP LOCATION 58
:3123          T59=59               : TEMP LOCATION 59
:3124
:3125          BEDB=5A              :UBE (BUS EXERCISOR) DATA BUF REG
:3126          BECC=5B              :UBE (BUS EXERCISOR) CYCLE COUNT REG
:3127          BEBA=5C              :UBE (BUS EXERCISOR) ADDR REG
:3128          BECR1=5D             :UBE (BUS EXERCISOR) CONTROL REG 1
:3129          CLEAR.ERR.ADDR=5E    :UBE CLEAR ERROR REG ADDRESS
:3130          BECR2=5F             :UBE (BUS EXERCISOR) CONTROL REG 2
:3131
:3132          #3(H)=60              : CONSTANT
:3133          #12(H)=61             : CONSTANT
:3134          #33333333=62          : CONSTANT
:3135          #0F0F0F0F=63          : CONSTANT
:3136          #00FF00FF=64          : CONSTANT
:3137          #162(H)=65            : CONSTANT FOR LS TRANSFER POINTER
:3138          BR7.IDENT=66          : BR7 IDENTIFIER (1010(B) IN BITS 5-2)
:3139          BG7=67               : BG7 ADDRESS (BITS 2 AND 3 SET)
:3140
:3141          :TEMPS FOR CPU TESTS
:3142          L30=30                :LS 30 TEMP (RESTORED TO 10000 AFTER USE)
:3143          L31=31                :LS 31 TEMP (RESTORE TO 20000 AFTER USE)
:3144          L53=53                :LS 53 TEMP
:3145          L73=73                :LS 73 TEMP
:3146
:3147          :REGISTER ADDRESSES
:3148
:3149          CONTROL.OS=78          : HARDWARE INDEX TO CONTROL WORDS (LS 40-4F)
:3150          SHIFT.OS(3-0)=79      : 16 LOCATION HARDWARE INDEX TO SHIFT PATTERN
:3151                                     (LS 20-2F)
:3152          SHIFT.OS(4-0)=7B      : 32 LOCATION HARDWARE INDEX TO SHIFT PATTERN
:3153                                     (LS 20-3F)
:3154          OS=7C                  : OS REGISTER
:3155          DEC.CON=7D             : DECIMAL CARRIES
:3156          SIZE=7E               : SIZE REGISTER
:3157          MDT= 7E               : MEMORY REFERENCE DATA SIZE
:3158          INTERRUPT.VEC=7E      : HARDWARE INTERRUPT VECTOR
:3159          :
:3160          : THIS WORD IS THE HARDWARE CONDITION CODES. THE CC'S CAN BE READ
:3161          : OR ARE WRITTEN HERE. NO OTHER BITS IN THE PSL ARE EFFECTED.
:3162          :
:3163          PSL.CC=7F              : PSL CONDITION CODES
:3164
:3165

```



```

:3166 .TOC "COMMON LS ASSIGNMENTS (TO REGISTERS)"
:3167
:3168 .UPPER HALF OF LS
:3169
:3170 XGPR.OS=0F8 : HARDWARE INDEX TO XGPRS
:3171 USER.INDEX(3-0)=0F9 : 16 LOCATION HARDWARE INDEX TO DIAGNOSTIC DATA
:3172 : AREA (LS LOCATIONS 0A0 THROUGH 0AF)
:3173 USER.INDEX(4-0)=0FB : 32 LOCATION HARDWARE INDEX TO DIAGNOSTIC DATA
:3174 : AREA (LS LOCATIONS 0A0 THROUGH 0BF)
:3175 CCR=0FC : CONSOLE READ REGISTER
:3176 TIMER=0FD : INTERVAL TIMER VALUE.
:3177 ALU.CC=0FE : ALU CONDITION CODES
:3178
:3179 THIS LOCATION IS A WRITE ONLY REGISTER. THE FOLLOWING BITS IN THE
:3180 PSL ARE AFFECTED:
:3181
:3182 COMPATIBILITY MODE - BIT<31>
:3183 CURRENT MODE - BITS<25:24>
:3184 INTERRUPT PRIORITY LEVEL (IPL) - BITS<20:16>
:3185 TRACE TRAP ENABLE (REALLY T OR TP) - BIT<4>
:3186 NEGATIVE CONDITION CODE - BIT<3>
:3187 ZERO CONDITION CODE - BIT<2>
:3188 OVERFLOW CONDITION CODE - BIT<1>
:3189 CARRY CONDITION CODE - BIT<0>
:3190
:3191 PSL.HW=OFF : HARDWARE PSL BITS.
:3192 .TOC "PROGRAM SPECIFIC LS ASSIGNMENTS (LS 80-F7)"
:3193
:3194 THESE ADDRESSES ARE ONLY ACCESSIBLE WITH THE MOV MICRO INSTRUCTION.
:3195 STARTING AT ADDRESS 80(H). THEY ARE NOT USED BY ALL SOFTWARE MODULES
:3196 BUT ARE SPECIFIC TO THIS ONE.
:3197
:3198
:3199
:3200 L90=90 : TEMP LOCATION
:3201 ALTER.MIX=91 : DATA: AAAA5555
:3202 HI.IPL=92 : BITS 16-20 SET (IPL=1F)
:3203 #FFFFFFFC=95 : FREQUENTLY USED MASK (BOTTOM BYTE)
:3204 #000F=96
:3205 #00F0=97
:3206 #0F00=98
:3207 #F000=99
:3208 #3C000=9A
:3209 #7FFF8000=9B
:3210 #540000=9C
:3211 #F00000=9D
:3212 #FC0000=9E
:3213 #3F003FFF=9F
:3214 #254=0A0
:3215
:3216 LB0=0B0 : TEMP LOCATION
:3217
:3218 LD0=0D0 : TEMP LOCATION
:3219
:3220 LF0=0F0 : TEMP LOCATION

```

:3221
:3222
:3223
:3224
:3225
:3226
:3227
:3228
:3229
:3230
:3231
:3232
:3233
:3234
:3235
:3236
:3237
:3238
:3239
:3240
:3241
:3242
:3243
:3244
:3245
:3246
:3247
:3248
:3249
:3250
:3251
:3252
:3253
:3254
:3255
:3256
:3257
:3258
:3259
:3260
:3261
:3262
:3263
:3264
:3265
:3266
:3267
:3268
:3269
:3270

page 'Microinstruction Formats'

The following is taken directly from the NEBULA Hardware Specification.

MICROINSTRUCTION FORMATS

1. BASIC

22	21			16	15					9	8	7	6	5	4		0
+-----+-----+-----+-----+-----+																	
1		DP						D ADRS				B		CC		SCTL	
+-----+-----+-----+-----+-----+																	

2. MOVE

22		20	19	17	16					9	8	7	6	5	4		0
+-----+-----+-----+-----+-----+																	
0		1		MDP				XD ADRS				B		CC		SCTL	
+-----+-----+-----+-----+-----+																	

3. EXTENDED

22		20	19			14	13	11		9	8	7	6	5	4		0
+-----+-----+-----+-----+-----+																	
0		1		0		XDP		SI		A		B		CC		SCTL	
+-----+-----+-----+-----+-----+																	

4. MEM.REQ

22			19	18	16	15				9	8	7	6	5	4		0
+-----+-----+-----+-----+-----+																	
0		0		1		MF1				D ADRS				MF2		DT	
																SCTL	
+-----+-----+-----+-----+-----+																	

5. MISC

22			19	18	16	15		12	11	9	8	7	6	5	4		0
+-----+-----+-----+-----+-----+																	
0		0		1		0		P		0		0		M1		M2	
										0		R		0		0	
																SCTL	
+-----+-----+-----+-----+-----+																	

6. JUMP

22			19	18	17										4	3	0	
+-----+-----+-----+-----+-----+																		
0		0		0		1		OS		JUMP ADDRESS								JCTL
+-----+-----+-----+-----+-----+																		

7. DECODE

22			19	18	17			12	11	9	8	7	6	5	4	3	0
+-----+-----+-----+-----+-----+																	
0		0		0		0		DEC.ADRS				IFUNC				JCTL	
+-----+-----+-----+-----+-----+																	

BACK UP PC				IB REQ---				---				OPC/SPEC							
SEL CM HI IR BYTE---								---								LOAD RDEST			
												LOAD OS---							

:3271 :page
:3272 :
:3273 :
:3274 :1. "BASIC" Microinstruction
:3275 :
:3276 :CSR<22> = 1 (OPCODE)
:3277 :
:3278 :This opcode bit causes LS Adrs <7> to be cleared.
:3279 :
:3280 :CSR<21:16> (Data Path Control)
:3281 :
:3282 :This field controls the data path. It controls the 2901 ALU,
:3283 :the ALU data source, the data destination in the 2901 array,
:3284 :and the write to the Local Store.
:3285 :
:3286 :CSR<15:9> (D Adrs <6:0>)
:3287 :
:3288 :This field selects which of the 128 accessible Local Store
:3289 :locations to use.
:3290 :
:3291 :CSR<8:7> (B Adrs)
:3292 :
:3293 :This field selects one of the four Working Registers.
:3294 :
:3295 :CSR<6:5> (Condition Codes)
:3296 :
:3297 :This field specifies the data type and condition code control.
:3298 :
:3299 :CSR<4:0> (SCTL)
:3300 :
:3301 :This field specifies the skip condition/micro sequencer
:3302 :function.

3303 .page
3304
3305
3306 2. "MOVE" Microinstruction
3307
3308 CSR<22:20> = 011 (OPCODE)
3309
3310 This combination of opcode bits allows CSR<16> to control
3311 LS Adrs <7>. This permits access of LS locations 80 - FF.
3312
3313 CSR<19:17> (Data Path Control)
3314
3315 This field is decoded to produce some data path control
3316 information.
3317
3318 0 LS[XD] <- WR[B], WR[B] <- MEM DATA* 4 LS[XD] <- MEM DATA*
3319 1 MEMORY <- LS [XD]* 5 LS[XD] <- ACC/PORT
3320 2 Q <- XWR[B] 6 LS[XD] <- WR[B] + OS
3321 3 WR[B] <- LS [XD] 7 LS[XD] <- WR[B]
3322
3323 CSR<16:9> (XD<7:0>)
3324
3325 This field specifies which of the 256 Local Store locations to
3326 use.
3327
3328 CSR<8:7> (B Adrs)
3329
3330 This field specifies which of the four Working Registers to
3331 use. For MDP = 2, this field selects either the Backup PC or
3332 the VAR.
3333
3334 CSR<6:5> (CC)
3335
3336 This field specifies the data type and condition code control.
3337
3338 CSR<4:0> (SCTL)
3339
3340 This field specifies the Skip control. See Table 3.
3341
3342
3343 * For information on which Working Registers and Local Store locations
3344 may be used for Memory Addresses and data source/destinations, see
3345 the Memory Management documentation. Note that these restrictions
3346 are due to microcoding conventions and not hardware.

:3347
:3348
:3349
:3350
:3351
:3352
:3353
:3354
:3355
:3356
:3357
:3358
:3359
:3360
:3361
:3362
:3363
:3364
:3365
:3366
:3367
:3368
:3369
:3370
:3371
:3372
:3373
:3374
:3375
:3376
:3377
:3378
:3379
:3380
:3381
:3382
:3383
:3384
:3385
:3386
:3387

.page

3. "EXTENDED" Microinstruction.

CSR<22:20> = 010 (OPCODE)

This opcode causes a function internal to the 2901 array; that is, an operation involving only Working Registers and the Q-Register.

CSR<19:14> (Data Path Control)

This field is decoded to supply the 2901 ALU function code, the ALU Source control, the destination code, and the ALU Carry-In.

CSR<13:11> (Shift Input)

This field selects the data to be shifted into a register, when a Shift function is being done.

CSR<10:9> (A Adrs)

This field selects which Working Register to use as a source of data, when RAM A is selected to the ALU.

CSR<8:7> (B Adrs)

This field selects which Working Register to use as a source (when RAM B is selected to the ALU) and/or destination (when the RAM is written) of data.

CSR<6:5> (CC)

This field specifies the data type and condition code control.

CSR<4:0> (SCTL)

This field specifies the skip condition/micro sequencer function.

:3388 :page
:3389 :
:3390 :
:3391 :4. 'MEM.REQ' Microinstruction
:3392 :
:3393 :CSR<22:19> = 0011 (OPCODE)
:3394 :
:3395 :This opcode causes a Memory Request to be started. The Local
:3396 :Store is output on the D-Bus, to be used as the virtual
:3397 :address. Working Register 5 is written with this address.
:3398 :
:3399 :CSR<18:16>, <8:7> (Memory Function)
:3400 :
:3401 :This field specifies to the Memory Controller the type of
:3402 :request: physical, virtual, type of access, etc. For the
:3403 :list of functions and the codes, see the NEBULA Memory Spec.
:3404 :
:3405 :CSR<15:9> (D Adrs <6:0>)
:3406 :
:3407 :This field selects which of the 128 accessible Local Store
:3408 :locations to use as the virtual address.
:3409 :
:3410 :CSR<6:5> (Data Type)
:3411 :
:3412 :This field specifies the data type of the request.
:3413 :
:3414 :00:=BYTE
:3415 :01:=WORD
:3416 :10:=SIZE Reg
:3417 :11:=LONG
:3418 :
:3419 :CSR<4:0> (SCTL)
:3420 :
:3421 :This field specifies the skip condition/micro sequencer
:3422 :function.

:3423 :page

:3424

:3425

:3426 :5. "MISC" Microinstruction

:3427

:3428 CSR<22:19> = 0010 (OPCODE)

:3429

:3430 This combination of opcode bits causes the data path to no-op,

:3431 and enables the MISC decoders.

:3432

:3433 CSR<18>

:3434 (Port/Misc Select)

:3435

:3436 This bit defines the instruction to be either a Misc or a Port

:3437 instruction. For CSR<18> = 1, CSR<17:10> is the Command Byte

:3438 to the Port, CSR<9:5> must be zero and CSR<4:0> is the SCTL

:3439 field. For CSR<18> = 0, it is a Misc instruction and the rest

:3440 of the word is defined as follows.

:3441

:3442 CSR<17:16>

:3443 (Not Used)

:3444

:3445 CSR<15:12> (Misc Function 1)

:3446

:3447 These four bits are decoded to do the following functions:

:3448

:3449 0 = CLEAR State Reg<1:0>

:3450 1 = CLEAR State Reg<1>, SET State Reg<0>

:3451 2 = SET State Reg<1>, CLEAR State Reg<0>

:3452 3 = CLEAR State Reg<0>

:3453 4 = CLEAR State Reg<1>

:3454 5 = SET State Reg<0>

:3455 6 = SET State Reg<1>

:3456 7 = SET REGISTER BACKUP MASK FLAG

:3457 8 = SET ACC I/O MODE

:3458 9 = SET PORT I/O MODE

:3459 A = CLEAR WCS HI ADRS

:3460 B = SET WCS HI ADRS

:3461 C = CLEAR CPU ATTN AND CPU ACK

:3462 D = SET CPU ACK

:3463 E = SET CPU ATTN

:3464 F = NOP

:3465

:3466 CSR<11:9> (Misc Function 2)

:3467

:3468 This field is decoded to do the following functions:

:3469

:3470 0 = NOP

:3471 1 = Mask IRD Interrupt Detection during Next State

:3472 2 = Assert CPU Data Available and pass WR to ACC/Port

:3473 3 = Trap ACC

:3474 4 = Read ACC uPC

:3475 5 = Assert XFER GRANT to Port

:3475 6 = Mask Halt and I-Bit Traps

:3475 7 = Write WR to Console Read Register

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) Microinstruction Fo 3-H 7
3-MAR-82 09:34:33 3-MAR-82
Microinstruction Formats

Fiche 1 Frame H7 Sequence 85
ENKCA Micro-diagnostic for DAP module Rev 01.00 Page 79

:3476 :page
:3477 :
:3478 : CSR<8> (MUST BE ZERO)
:3479 :
:3480 : CSR<7> (WR Address)
:3481 :
:3482 :
:3483 : This bit selects WR[0] or WR[1] to be put on the Y-Bus.
:3484 :
:3485 : CSR<6:5> (Not Used)
:3486 :
:3487 : CSR<4:0> (SCTL)
:3488 :
:3489 : This field specifies the skip condition/micro sequencer
:3490 : function.
:3491 :

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) Microinstruction Fo 3-MAR-82
3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
Microinstruction Formats

Fiche 1 Frame 17

Sequence 86
Rev 01.00

Page 80

:3492 :.page
:3493 :
:3494 :
:3495 :6. "JUMP" Microinstruction
:3496 :
:3497 :CSR<22:19> = 0001 (OPCODE)
:3498 :
:3499 :This opcode causes the data path to no-op, and the micro
:3500 :sequencer to execute a JUMP.
:3501 :
:3502 :CSR<18> (Select OS)
:3503 :
:3504 :This bit, when asserted, causes OS<4:0> to be OR'ed with the
:3505 :five low bits of the of the jump microaddress.
:3506 :
:3507 :CSR<17:4> (Jump Microaddress)
:3508 :
:3509 :If Select OS is CLEAR, this field specifies the fourteen-bit
:3510 :jump micro address. If Select OS is SET, CSR<17:9> is Jump
:3511 :Adrs<13:5> and OS<4:0> OR'ed with CSR<8:4> is Jump Adrs<4:0>.
:3512 :
:3513 :CSR<3:0> (JCTL)
:3514 :
:3515 :This field specifies the jump condition/micro sequencer
:3516 :function.

:3517 :page

:3518
:3519
:3520
:3521
:3522
:3523
:3524
:3525
:3526
:3527
:3528
:3529
:3530
:3531
:3532
:3533
:3534
:3535
:3536
:3537
:3538
:3539
:3540
:3541
:3542
:3543
:3544
:3545
:3546
:3547
:3548
:3549
:3550
:3551
:3552
:3553
:3554
:3555
:3556
:3557
:3558
:3559
:3560
:3561
:3562
:3563
:3564
:3565
:3566
:3567

7. "DECODE" Microinstruction

A. DESCRIPTION

CSR<22:19> = 0000 (OPCODE)

This combination of opcode bits causes the Local Store to access location 10 (the PC.) The Local Store drives the D-Bus. The 2901 is set up to input the data on the D-Bus, increment it, and output to the Y-Bus. Thus, the Y-Bus contains the PC + 1.

CSR<18> (BPC [asserted low])

If this bit is CLEAR, the incremented PC is written into the 2901 RAM. If it is SET, the RAM is not written.

CSR<17:12> (DEC.ADRS <13:8>)

This data is taken to be the upper six bits of the next microaddress. The low eight bits are supplied by the IRD ROM. The actual JUMP/JSR/NO-OP is determined by the JCTL field.

CSR<11:9> (IFUNC)

This is the IFUNC field. It defines which fork is being taken.

CSR<8> (IB.REQ)

This is the IB.REQ bit. When it is CLEAR, the LS is not written, no interaction with memory is initiated and the instruction buffer is not affected.

When it is SET:

The incremented PC (valid on the Y-Bus by the end of the cycle) is re-written to Local Store location 0. This completes the PC increment function.

A Conditional Memory Request is executed. If the two low bits of the PC are not 11, the request is not initiated. If the two low bits are set, an IB PREFETCH is done. The (unincremented) PC is output to the memory and the CPU grant (CPUG) signal is examined. If CPUG is low by T120, the CPU will stall until the CPUG signal becomes true. After the request is completed, the next state entered is dependent upon the JCTL field.

:3568 :page
:3569 :
:3570 :
:3571 :
:3572 :
:3573 :
:3574 :
:3575 :
:3576 :
:3577 :
:3578 :
:3579 :
:3580 :
:3581 :
:3582 :
:3583 :
:3584 :
:3585 :
:3586 :
:3587 :
:3588 :
:3589 :
:3590 :
:3591 :
:3592 :
:3593 :
:3594 :
:3595 :
:3596 :
:3597 :
:3598 :
:3599 :
:3600 :
:3601 :
:3602 :
:3603 :

CSR<7> (SEL CM HI IR BYTE)

This bit enables the upper byte of the Compatibility Mode instruction in the Prefetch register to drive the IB-Bus. It is used only while in Compatibility Mode, when IRD is being done. It must never be asserted in VAX Mode.

CSR<6> (LD.OS)

If this bit is SET, the eight-bit data on the IB Bus (the output of the instruction buffer) is loaded into the OS register. This load is dependent upon Compatibility Mode and LD R Dest (If CM.AND.LD R Dest, OS<3> <- 0.) If the bit is CLEAR, OS is not affected.

CSR<5> (LD R DEST)

If this bit is SET, the R Dest Skip bit will be SET for Specifier Mode equal to 5 and CLEARED for Mode not equal to 5, in VAX mode, or mode 0 while in Compatibility Mode. If this bit is CLEAR, the R Dest bit will be unaffected.

CSR<4> (OPC/SPEC)

This bit (effectively another IFUNC bit) defines the data to be decoded; that is, if it is an Opcode or a Specifier. In hardware, it selects which ROM is to drive the micro address bus.

CSR<3:0> (JCTL)

These four bits are decoded to make one of 16 functions. See Table 3.

:3604
:3605
:3606
:3607
:3608
:3609
:3610
:3611
:3612
:3613
:3614
:3615
:3616
:3617
:3618
:3619
:3620
:3621
:3622
:3623
:3624
:3625
:3626
:3627
:3628
:3629
:3630
:3631
:3632
:3633
:3634
:3635
:3636
:3637
:3638
:3639
:3640
:3641
:3642
:3643
:3644
:3645
:3646
:3647
:3648
:3649

page "Tables"

TABLES

Table 1. 2901 Control Decoding

The 2901 control decoder examines CSR<22:14>. This nine-bit field of the microword changes in meaning for the different cases of micro opcode.

	CSR	22	21	20	19	18	17	16	15	14	
		--	--	--	--	--	--	--	--	--	
1. BASIC		1	[Six bit function code]						X	X	
2. MOVE		0	1	1	[function]				X	X	X
3. EXTENDED		0	1	0	[Six bit function code]						
4. MEM.REQ		0	0	1	1	X	X	X	X	X	WR[5] <- D
5. MISC		0	0	1	0	X	X	X	X	X	Y-Bus <- WR
6. JUMP		0	0	0	1	X	X	X	X	X	NO-OP
7. DECODE		0	0	0	0	BPC*	X	X	X	X	

If BPC = 0, Write WR; else, No-op

This decoder is implemented using a ROM and a PAL. 512 locations are needed for CSR<22:14> to index into. The decoding logic supplies 10 control bits:

ALU Source Select (3)
ALU Function (3)
ALU Result Destination (3)
ALU Carry-In (1)

The ROM addresses are therefore allocated in the following way:

Location	0 - F	Incr. D-Bus, Output ALU, No Write.
	10 - 1F	Incr. D-Bus, Output ALU, Write RAM.
	20 - 3F	No-Op.
	40 - 5F	Output WR, bypassing ALU.
	60 - 7F	Pass D-Bus through ALU and write RAM.
	80 - BF	CSR<19:14> indicates 2901 function.
	C0 - FF	CSR<19:17> indicates 2901 function.
	100 - 1FF	CSR<21:16> indicates 2901 function.

For the detailed function table see the Microcode Listing.

:3650
:3651
:3652
:3653
:3654
:3655
:3656
:3657
:3658
:3659
:3660
:3661
:3662
:3663
:3664
:3665
:3666
:3667
:3668
:3669
:3670
:3671
:3672
:3673
:3674
:3675
:3676
:3677
:3678
:3679
:3680
:3681
:3682
:3683
:3684
:3685
:3686
:3687
:3688
:3689

page

Table 2. Local Store Write

The Local Store Write Controller examines CSR<22:17>, CSR<6>, the SIZE register (SIZE<1:0>) and the Compatibility Mode bit (CM). These eight bits are decoded to determine which parts of the LS to write, if any. The LS is written with data from the 2901 ALU (Y-Bus).

CSR	22	21	20	19	18	17	
1. BASIC	1	0	X	X	X	X	No Write
	1	1	X	X	X	X	Write *
2. MOVE	0	1	1	0	X	1	No Write
	0	1	1	0	1	X	No Write
	0	1	1	1	X	X	Write *
3. EXTENDED	0	1	1	X	X	X	No Write
4. MEM.REQ	0	0	1	1	X	X	No Write
5. MISC	0	0	1	0	X	X	No Write
6. JUMP	0	0	0	1	X	X	No Write
7. DECODE	0	0	0	0	X	X	Conditional Write **

* LS Write, Data Type Dependent. These cases now examine CSR<6>.

If CSR<6> = 0, and CM = 0 Write LS<31:0> (Data Type = LONG)
If CSR<6> = 0, and CM = 1 Write LS<15:0> (Compatibility Mode. DT = WORD)

If CSR<6> = 1, the Write is conditional on the SIZE register.

If CM = 0 SIZE<1:0> = 00 Write LS<7:0> (BYTE)
= 01 Write LS<15:0> (WORD)
= 1X Write LS<31:0> (LONG)

If CM = 1 Write LS<15:0> (WORD)

** If CSR<8> = 0, No Write
If CSR<8> = 1, Write LS<31:0> (Writes incremented PC)

:3690
:3691
:3692
:3693
:3694
:3695
:3696
:3697
:3698
:3699
:3700
:3701
:3702
:3703
:3704
:3705
:3706
:3707
:3708
:3709
:3710
:3711
:3712
:3713
:3714
:3715
:3716
:3717
:3718
:3719
:3720
:3721
:3722
:3723
:3724
:3725
:3726
:3727
:3728
:3729
:3730
:3731
:3732
:3733
:3734

page

Table 3. SKIP/JUMP Conditions

This table summarizes the Skip and Jump conditions, and the other special functions of the micro sequencer.

SCTL codes 0-F, 17-1F are Skip conditions. The codes 10-17 execute special functions. JCTL codes 0-B are Jump conditions, and C-F execute special functions.

Sctl	Function	Sctl	Function
00	Not ALU N	10	RTN+1 if No Mem Err
01	Not ALU Z	11	Loop if Not ALU Z
02	Not ALU V	12	POP Stack
03	ALU C	13	Loop if ALU C
04	Not PSL C	14	Return
05	Branch False	15	No Skip (NOP)
06	RDEST	16	RTN+1
07	Not Inter. Req.	17	Loop if No Inter. and No Sync
08	ALU N	18	Console ATTN
09	ALU Z	19	Console ACK
0A	ALU V	1A	Port Interrupt Pending
0B	Not ALU C	1B	Reg. Backup Mask Flag
0C	State 0	1C	ALU N.XOR.ALU V
0D	State 1	1D	Not Memory Error
0E	Not State 0	1E	Skip
0F	Not State 1	1F	Sync

Jctl	Function	Jctl	Function
00	Not ALU N	08	ALU N
01	Not ALU Z	09	ALU Z
02	Not ALU V	0A	ALU V
03	ALU C	0B	Not ALU C
04	JUMP	0C	JSR
05	Branch False	0D	JSR if IB Valid
06	RDEST	0E	No Jump; Skip if IB Valid
07	Not Interr. Req	0F	IB Valid

:3735 .PAGE "2901 ALU Control Description"
:3736
:3737
:3738
:3739
:3740
:3741
:3742
:3743
:3744
:3745
:3746
:3747
:3748
:3749
:3750
:3751
:3752
:3753
:3754
:3755
:3756
:3757
:3758
:3759
:3760
:3761
:3762
:3763
:3764
:3765
:3766
:3767
:3768
:3769
:3770
:3771
:3772
:3773
:3774
:3775
:3776
:3777
:3778
:3779
:3780
:3781
:3782

The 2901 ALU is the heart of the CPU module. It is responsible for all CPU calculations. The control lines going to 10 through 18 control the function, input, and output of the 2901. When a failure involving the 2901 occurs, the logical place to start looking is the control lines. The following tables describe what the control signals should be for various functions. Use it whenever the control lines are to be checked. The test descriptions tell the operator what function the 2901 is supposed to be doing.

The SRC CTL on lines 10 through 12 (pins 12 through 14 respectively) control where the inputs to the internal ALU within the 2901 come from. The inputs to the internal ALU are labeled R and S. R is one operand (4 bits) and S is the other operand (also 4 bits). The R input can come from a Working Register (labeled A), the D bus (direct input), or be forced to 0. The S input can come from either set of working registers (labeled A and B), the Q register, or forced to 0. A table of these control signals follow:

			octal code	SRC operands	
12	11	10		R	S
L	L	L	0	A	Q
L	L	H	1	A	B
L	H	L	2	0	Q
L	H	H	3	0	B
H	L	L	4	0	A
H	L	H	5	D	A
H	H	L	6	D	Q
H	H	H	7	D	0

ALU SOURCE OPERAND CONTROL

The ALU CTL on lines 13 through 15 (pins 26, 28, 27 respectively) control what function the ALU is to perform on the two operands R and S. A table follows:

3783 page
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836

15	14	13	octal code	ALU function
L	L	L	0	R PLUS S
L	L	H	1	S MINUS R
L	H	L	2	R MINUS S
L	H	H	3	R OR S
H	L	L	4	R AND S
H	L	H	5	R NOT AND S
H	H	L	6	R XOR S
H	H	H	7	R XNOR S

ALU FUNCTION CONTROL

The DST CTL on lines 16 through 18 (pins 5, 7, 6 respectively) control what goes out onto the Y bus (either the internal ALU result or a WR directly) and what happens internal to the 2901 (shifting, writing the Q register etc). Internal shifting left or right is accomplished with this control. Below is a table explaining the various destination possibilities. The arrow refers to where the output will end up. B indicates the WR addressed by the B ADDR, F refers to the output of the internal ALU, and A refers to the output of the WR addressed by the A ADDR. Up is toward the MSB, while down is toward the LSB.

nomenclature in listing	18	17	16	octal code	RAM function shift	load	Q-reg function shift	load	Y output
LOAD Q	L	L	L	0	NONE	NONE	NONE	F->Q	F
NOP	L	L	H	1	NONE	NONE	NONE	NONE	F
WRITE.B.A	L	H	L	2	NONE	F->B	NONE	NONE	A
WRITE.B.F	L	H	H	3	NONE	F->B	NONE	NONE	F
RSHF.RAM.Q	H	L	L	4	DOWN	F/2->B	DOWN	Q/2->Q	F
RSHF.RAM	H	L	H	5	DOWN	F/2->B	NONE	NONE	F
LSHF.RAM.Q	H	H	L	6	UP	2F->B	UP	2Q->Q	F
LSHF.RAM	H	H	H	7	UP	2F->B	NONE	NONE	F

ALU DESTINATION CONTROL

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

2901 ALU Control De 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
2901 ALU Control Description

Fiche 1 Frame D8

Sequence 94

Page 88

```
:3837 :page  
:3838 :TOC "DIAGNOSTIC MACROS"  
:3839  
:3840 LS.DATA.WORD/=<15:0>  
:3841 UPPER.BYTE/=<23:16>  
:3842  
:3843 WORD[] "UPPER.BYTE/0,LS.DATA.WORD/@1"  
:3844 COUNT.LS[] "UPPER.BYTE/0,LS.DATA.WORD/@1"  
:3845 COUNT.MM[] "UPPER.BYTE/1,LS.DATA.WORD/@1"  
:3846 START.ADR[] "UPPER.BYTE/0,LS.DATA.WORD/@1"  
:3847  
:3848 ASCII.LIT/=<15:0>  
:3849  
:3850 CA=4341 ;ASCII VALUE FOR FILE NAMES  
:3851 CB=4342 :  
:3852 CC=4343 :  
:3853 CD=4344 :  
:3854 CE=4345 : V  
:3855 CF=4346  
:3856 CG=4347  
:3857 CH=4348  
:3858  
:3859 ASCII [] "UPPER.BYTE/0,ASCII.LIT/@1"  
:3860  
:3861 .REGION/0,6F
```

			:3862
			:3863
			:3864
			:3865
			:3866
			:3867
			:3868
			:3869
			:3870
			:3871
			:3872
U	0001,	0871,B4	:3873
U	0002,	8876,B4	:3874
U	0003,	087D,24	:3875
U	0004,	0981,04	:3876
U	0005,	098D,F4	:3877
U	0006,	8997,F4	:3878
U	0007,	09A1,74	:3879
U	0008,	89AA,B4	:3880
U	0009,	09B5,84	:3881
U	000A,	89B9,94	:3882
U	000B,	09D9,14	:3883
U	000C,	09DD,C4	:3884
U	000D,	09E2,64	:3885
U	000E,	09E8,04	:3886
U	000F,	89EB,74	:3887
U	0010,	09ED,04	:3888
U	0011,	89EE,B4	:3889
U	0012,	89EF,C4	:3890
U	0013,	89F1,34	:3891
U	0014,	0B02,04	:3892
U	0015,	8B0F,C4	:3893
U	0016,	0B13,04	:3894
U	0017,	8B1C,B4	:3895
U	0018,	0B2A,F4	:3896
U	0019,	8B35,34	:3897
U	001A,	8B3A,54	:3898
U	001B,	8B41,94	:3899
U	001C,	8B44,A4	:3900
U	001D,	8B49,E4	:3901
U	001E,	8B53,64	:3902
U	001F,	8B59,94	:3903
U	0020,	8B5F,C4	:3904
U	0021,	886B,E4	:3905
			:3906
U	0022,	886B,E4	:3907
			:3908
U	0023,	886B,E4	:3909
			:3910
U	0024,	886B,E4	:3911
			:3912
U	0025,	886B,E4	:3913
			:3914
U	0026,	886B,E4	:3915
			:3916

.PAGE "TEST POINTERS"

This portion contains the pointers to all tests in this section. The WCS address of the JUMP instruction corresponds to the test number. E.g. WCS 5 contains a JUMP to test 5. All unused test numbers will contain a JUMP to an error routine. This portion is 63. locations long, allowing for up to 63. tests per section, from WCS address 1 to WCS address 3F.

:TEST POINTERS BEGIN AT WCS ADDRESS 1

```

JMP [T.1]
JMP [T.2]
JMP [T.3]
JMP [T.4]
JMP [T.5]
JMP [T.6]
JMP [T.7]
JMP [T.8]
JMP [T.9]
JMP [T.A]
JMP [T.B]
JMP [T.C]
JMP [T.D]
JMP [T.E]
JMP [T.F]
JMP [T.10]
JMP [T.11]
JMP [T.12]
JMP [T.13]
JMP [T.14]
JMP [T.15]
JMP [T.16]
JMP [T.17]
JMP [T.18]
JMP [T.19]
JMP [T.1A]
JMP [T.1B]
JMP [T.1C]
JMP [T.1D]
JMP [T.1E]
JMP [T.1F]
JMP [T.20]
JMP [ERR.NO.TEST]
JMP [ERR.NO.TEST]
JMP [ERR.NO.TEST]
JMP [ERR.NO.TEST]
JMP [ERR.NO.TEST]
JMP [ERR.NO.TEST]

```

[illegible]

U 0027, 886B,E4	:3917	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3918		:IN THIS OVERLAY
U 0028, 886B,E4	:3919	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3920		:IN THIS OVERLAY
U 0029, 886B,E4	:3921	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3922		:IN THIS OVERLAY
U 002A, 886B,E4	:3923	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3924		:IN THIS OVERLAY
U 002B, 886B,E4	:3925	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3926		:IN THIS OVERLAY
U 002C, 886B,E4	:3927	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3928		:IN THIS OVERLAY
U 002D, 886B,E4	:3929	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3930		:IN THIS OVERLAY
U 002E, 886B,E4	:3931	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3932		:IN THIS OVERLAY
U 002F, 886B,E4	:3933	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3934		:IN THIS OVERLAY
U 0030, 886B,E4	:3935	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3936		:IN THIS OVERLAY
U 0031, 886B,E4	:3937	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3938		:IN THIS OVERLAY
U 0032, 886B,E4	:3939	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3940		:IN THIS OVERLAY
U 0033, 886B,E4	:3941	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3942		:IN THIS OVERLAY
U 0034, 886B,E4	:3943	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3944		:IN THIS OVERLAY
U 0035, 886B,E4	:3945	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3946		:IN THIS OVERLAY
U 0036, 886B,E4	:3947	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3948		:IN THIS OVERLAY
U 0037, 886B,E4	:3949	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3950		:IN THIS OVERLAY
U 0038, 886B,E4	:3951	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3952		:IN THIS OVERLAY
U 0039, 886B,E4	:3953	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3954		:IN THIS OVERLAY
U 003A, 886B,E4	:3955	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3956		:IN THIS OVERLAY
U 003B, 886B,E4	:3957	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3958		:IN THIS OVERLAY
U 003C, 886B,E4	:3959	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3960		:IN THIS OVERLAY
U 003D, 886B,E4	:3961	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3962		:IN THIS OVERLAY
U 003E, 886B,E4	:3963	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3964		:IN THIS OVERLAY
U 003F, 886B,E4	:3965	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3966		:IN THIS OVERLAY
U 0040, 886B,E4	:3967	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3968		:IN THIS OVERLAY
U 0041, 886B,E4	:3969	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
	:3970		:IN THIS OVERLAY
U 0042, 886B,E4	:3971	JMP [ERR.NO.TEST]	:GO TO ERROR ROUTINE. NO SUCH TEST NUMBER

U 0043, 886B,E4	:3972
	:3973
U 0044, 886B,E4	:3974
	:3975
U 0045, 886B,E4	:3976
	:3977
U 0046, 886B,E4	:3978
	:3979
U 0047, 886B,E4	:3980
	:3981
U 0048, 886B,E4	:3982
	:3983
U 0049, 886B,E4	:3984
	:3985
U 004A, 886B,E4	:3986
	:3987
U 004B, 886B,E4	:3988
	:3989
U 004C, 886B,E4	:3990
	:3991
U 004D, 886B,E4	:3992
	:3993
U 004E, 886B,E4	:3994
	:3995
U 004F, 886B,E4	:3996
	:3997
	:3998

[illegible][illegible]

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC POINTERS

DIAGNOSTIC POINTERS 3-MAR-82

Fiche 1 Frame H8

Sequence 98

Rev 01.00

Page 92

```
:3999 .PAGE "DIAGNOSTIC POINTERS"
:4000
:4001 This section contains all the pointers needed for this diagnostic
:4002 overlay. The first pointer (at WCS location 0) points to a data
:4003 transfer routine. It is the first location executed after a new
:4004 WCS load. The instruction here is a JUMP to TRANSFER.POINT which may
:4005 contain instructions to transfer data. If no data is to be trans-
:4006 ferred, or at the completion of the data transfers, the CPU will
:4007 issue CPU ATTN with all bits of the control and status word clear.
:4008
:4009 The next 80. locations (from WCS location 1 to WCS location 4F)
:4010 contain pointers to each test in this overlay. The pointers are
:4011 JUMP instructions to the test number corresponding to the location
:4012 number. E.g. location 5 will contain a JUMP to test 5. All unused
:4013 test numbers will contain a JUMP to an error routine. This portion
:4014 appears after the definitions in this listing.
:4015
:4016 Finally the next 32. locations (from WCS location 50 to 6F) contain
:4017 pointers (Jump instructions) to WCS subroutines which are to be used by
:4018 the console diagnostic monitor.
:4019
U 0000, 086C.24 :4020 0: JMP [TRANSFER.POINT] :() TO ROUTINE THAT LOADS LS 7 WITH
:4021 :() POINTER TO DATA SECTION AND ISSUES
:4022 : CPU ATTN WITH TRANSFER BIT SET.
:4023
:4024 50: :START AT WCS LOCATION 50 FOR SUB-
:4025 : ROUTINE POINTERS
:4026
:4027 : SUBROUTINE POINTERS
:4028
U 0050, 0860.84 :4029 JMP [DEPOSIT.CSR] :GO TO WCS SUBROUTINE WHICH WRITES THE
:4030 : MCT CSR FOR DEPOSIT CSR COMMAND
U 0051, 0860.D4 :4031 JMP [EXAMINE.CSR] :GO TO WCS SUBROUTINE WHICH READS THE
:4032 : MCT CSR FOR EXAMINE CSR COMMAND
U 0052, 0861.F4 :4033 JMP [DEPOSIT.TB] :GO TO WCS SUBROUTINE WHICH WRITES THE
:4034 : MCT TRANSLATION BUFFER FOR DEPOSIT
:4035 : TB COMMAND
U 0053, 8863.A4 :4036 JMP [EXAMINE.TB] :GO TO WCS SUBROUTINE WHICH READS THE
:4037 : MCT TRANSLATION BUFFER FOR EXAMINE
:4038 : TB COMMAND
U 0054, 8865.C4 :4039 JMP [DEPOSIT.MM] :GO TO WCS SUBROUTINE WHICH WRITES MAIN
:4040 : MEMORY FOR DEPOSIT MM COMMAND. ALSO
:4041 : USED TRANSFERRING DATA FROM WCS TO
:4042 : MAIN MEMORY.
U 0055, 0866.24 :4043 JMP [EXAMINE.MM] :GO TO WCS SUBROUTINE WHICH READS MAIN
:4044 : MEMORY FOR EXAMINE MM COMMAND.
U 0056, 0868.94 :4045 JMP [SAVE.WR] :GO TO WCS SUBROUTINE WHICH STORES THE
:4046 : CONTENTS OF WR0-WR3 IN LS 0-3
U 0057, 8868.E4 :4047 JMP [RESTORE.WR] :GO TO WCS SUBROUTINE WHICH RESTORES THE
:4048 : CONTENTS OF WR0-WR3 FROM LS 0-3
U 0058, 8864.24 :4049 JMP [DEPOSIT.UBS] :GO TO WCS SUBROUTINE WHICH WRITES THE
:4050 : UNIBUS MAP ON THE MEMORY CONTROLLER
U 0059, 0865.44 :4051 JMP [EXAMINE.UBS] :GO TO WCS SUBROUTINE WHICH READS THE
:4052 : UNIBUS MAP ON THE MEMORY CONTROLLER
U 005A, 0866.84 :4053 JMP [EXAMINE.ICSR] :GO TO WCS SUBROUTINE WHICH READS THE
```


ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC POINTERS

DIAGNOSTIC POINTERS 3-MAR-82

Fiche 1 Frame 18

Sequence 99

ENKCA Micro-diagnostic for DAP module Rev 01.00 Page 93

U 005B, 0866.B4	:4054	JMP [EXAMINE.IDAR]	: IDC CSR
U 005C, 0866.E4	:4055	JMP [EXAMINE.DBUF]	: GO TO WCS SUBROUTINE WHICH READS THE
U 005D, 8867.14	:4056	JMP [EXAMINE.PATT]	: IDC DAR
U 005E, 8867.44	:4057	JMP [EXAMINE.POSIT]	: GO TO WCS SUBROUTINE WHICH READS THE
U 005F, 0867.94	:4058	JMP [DEPOSIT.ICSR]	: IDC DBUF
U 0060, 0867.C4	:4059	JMP [DEPOSIT.IDAR]	: GO TO WCS SUBROUTINE WHICH READS THE
U 0061, 0867.F4	:4060	JMP [DEPOSIT.DBUF]	: IDC ECC PATTERN
U 0062, 8868.24	:4061	JMP [CLEAR.FIFO.ADDR]	: GO TO WCS SUBROUTINE WHICH READS THE
U 0063, 8868.44	:4062	JMP [SELECT.FIFO.A]	: IDC ECC POSITION
U 0064, 0868.64	:4063	JMP [SELECT.FIFO.B]	: GO TO WCS SUBROUTINE WHICH WRITES THE
U 0065, DB00.15	:4064	NOP	: IDC CSR
U 0066, DB00.15	:4065	NOP	: GO TO WCS SUBROUTINE WHICH WRITES THE
U 0067, DB00.15	:4066	NOP	: IDC DAR
U 0068, DB00.15	:4067	NOP	: GO TO WCS SUBROUTINE WHICH WRITES THE
U 0069, DB00.15	:4068	NOP	: IDC DBUF
U 006A, DB00.15	:4069	NOP	: GO TO THE WCS SUBROUTINE WHICH CLEARS
U 006B, DB00.15	:4070	NOP	: THE IDC FIFO ADDRESS
U 006C, DB00.15	:4071	NOP	: GO TO THE WCS SUBROUTINE WHICH SELECTS
U 006D, DB00.15	:4072	NOP	: FIFO A
U 006E, DB00.15	:4073	NOP	: GO TO THE WCS SUBROUTINE WHICH SELECTS
U 006F, DB00.15	:4074	NOP	: FIFO B
	:4075	NOP	: NOT USED
	:4076	NOP	: NOT USED
	:4077	NOP	: NOT USED
	:4078	NOP	: NOT USED
	:4079	NOP	: NOT USED
	:4080	NOP	: NOT USED
	:4081	NOP	: NOT USED
	:4082	NOP	: NOT USED
	:4083	NOP	: NOT USED
	:4084	NOP	: NOT USED
	:4085	NOP	: NOT USED
	:4086	NOP	: NOT USED

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) 3-MAR-82 09:34:33
DIAGNOSTIC POINTERS

DIAGNOSTIC POINTERS 3-MAR-82

Fiche 1 Frame J8

Sequence 100 Page 94
Rev 01.00

```
:4087 .PAGE
:4088 .REGION/70,5FF
:4089 .TOC "LS DATA SECTION"
:4090
:4091 :+++
:4092 This section lists the data that will be transferred to LS by the con-
:4093 sole as part of the initialization performed in test 0. The TRANSFER
:4094 DATA routine will point to this data area. LS contains the constants,
:4095 control and status word, and temporary storage locations used by the
:4096 tests and subroutines of the microdiagnostic.
:4097 The LS is 32 bits wide. Each of these words is 16 bits, so two con-
:4098 secutive words listed here will be loaded into each consecutive LS loc-
:4099 ation. The first word listed will be bits 0-15 of the LS location, the
:4100 second word listed will be bits 16-31 of the same LS location.
:4101 The locations 78-7F in the first half of LS and F8-FF in the second
:4102 half of LS are not actual LS locations. They force an access to one of
:4103 several registers in the CPU or force an indexing feature to another LS
:4104 location. For that reason, they are not written via the transfer rou-
:4105 tine.
:4106 Note: This table is in hexadecimal format i.e. each digit represents
:4107 four bits, so four digits represents a full 16 bit word. The micro-
:4108 assembler requires that a hexadecimal value that starts with a letter
:4109 (e.g. FFFF) be preceded by a 0 (0FFFF). So some table values will
:4110 contain 5 hexadecimal digits. The fifth digit is not actually used.
:4111 :---
:4112
:4113 TRANSFER.DATA.LS1:
:4114
:4115 U 0070, 0000,78 :4116 COUNT.LS[0078] :LONGWORD COUNT (NUMBER OF LONGWORDS TO TRANS-
:4117 :FER TO LS = 120 DECIMAL) DESTINATION IS LS
:4118 U 0071, 0000,00 :4119 START.ADR[0000] :DESTINATION START ADDRESS (START AT LS 0)
:4120 U 0072, 0000,00 :4121 WORD[0000] :LOCATION 0
:4122 U 0073, 0000,00 :4123 WORD[0000] :LOCATION 1
:4124 U 0074, 0000,00 :4125 WORD[0000] :LOCATION 2
:4126 U 0075, 0000,00 :4127 WORD[0000] :LOCATION 3
:4128 U 0076, 0000,00 :4129 WORD[0000] :LOCATION 4
:4130 U 0077, 0000,00 :4131 WORD[0000] :LOCATION 5
:4132 U 007A, 0000,00 :4133 WORD[0000] :LOCATION 6
:4134 U 007B, 0000,00 :4135 WORD[0000] :LOCATION 7
:4136 U 007C, 0000,00 :4137 WORD[0000]
:4138 U 007D, 0000,00 :4139 WORD[0000]
:4140 U 007E, 0000,00 :4141 WORD[0000]
:4141 U 007F, 0000,00 :4142 WORD[0000]
:4142 U 0080, 0000,00 :4143 WORD[0000]
```

Z7-ENKCA-1.0	:	ENKCA.MIC								
: ENKCA.MCR			MICRO2 1L(03)	LS DATA SECTION	3-MAR-82	09:34:33	3-MAR-82	Fiche 1 Frame K8	Sequence 101	Page 95
: ENKCA.MIC			LS DATA SECTION					ENKCA Micro-diagnostic for DAP module	Rev 01.00	

U 0081, 0000,00	:4142	WORD[0000]	
U 0082, 0000,00	:4143	WORD[0000]	:LOCATION 8
U 0083, 0000,00	:4144	WORD[0000]	
	:4145	WORD[0000]	
	:4146		
U 0084, 0000,00	:4147	WORD[0000]	:LOCATION 9
U 0085, 0000,00	:4148	WORD[0000]	
	:4149		
U 0086, 0000,00	:4150	WORD[0000]	:LOCATION 0A
U 0087, 0000,00	:4151	WORD[0000]	
	:4152		
U 0088, 0000,00	:4153	WORD[0000]	:LOCATION 0B
U 0089, 0000,00	:4154	WORD[0000]	
	:4155		
U 008A, 0000,00	:4156	WORD[0000]	:LOCATION 0C
U 008B, 0000,00	:4157	WORD[0000]	
	:4158		
U 008C, 0000,00	:4159	WORD[0000]	:LOCATION 0D
U 008D, 0000,00	:4160	WORD[0000]	
	:4161		
U 008E, 0000,00	:4162	WORD[0000]	:LOCATION 0E
U 008F, 0000,00	:4163	WORD[0000]	
	:4164		
U 0090, 0000,00	:4165	WORD[0000]	:LOCATION 0F
U 0091, 0000,00	:4166	WORD[0000]	
	:4167		
U 0092, 0000,00	:4168	WORD[0000]	:LOCATION 10
U 0093, 0000,00	:4169	WORD[0000]	
	:4170		
U 0094, 0000,00	:4171	WORD[0000]	:LOCATION 11
U 0095, 0000,00	:4172	WORD[0000]	
	:4173		
U 0096, 0000,00	:4174	WORD[0000]	:LOCATION 12
U 0097, 0000,00	:4175	WORD[0000]	
	:4176		
U 0098, 0000,FF	:4177	WORD[00FF]	:LOCATION 13
U 0099, 0000,00	:4178	WORD[0000]	
	:4179		
U 009A, 00FF,FF	:4180	WORD[00FFF]	:LOCATION 14
U 009B, 0000,00	:4181	WORD[0000]	
	:4182		
U 009C, 0000,00	:4183	WORD[0000]	:LOCATION 15
U 009D, 00FF,00	:4184	WORD[00FF00]	
	:4185		
U 009E, 00FF,00	:4186	WORD[00FF00]	:LOCATION 16
U 009F, 00FF,FF	:4187	WORD[00FFFF]	
	:4188		
U 00A0, 003F,FF	:4189	WORD[3FFFF]	:LOCATION 17
U 00A1, 0001,00	:4190	WORD[0100]	
	:4191		
U 00A2, 003F,FF	:4192	WORD[3FFFF]	:LOCATION 18
U 00A3, 00FF,FE	:4193	WORD[00FFFE]	
	:4194		
U 00A4, 00FF,FF	:4195	WORD[00FFFF]	:LOCATION 19
U 00A5, 00FE,7F	:4196	WORD[00FE7F]	

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03)
LS DATA SECTION

LS DATA SECTION
3-MAR-82 09:34:33

L 8

3-MAR-82

Fiche 1 Frame L8

ENKCA Micro-diagnostic for DAP module

Sequence 102
Rev 01.00

Page 96

U 00A6, 0000,00	:4197	WORD[0000]	:LOCATION 1A
U 00A7, 007F,F8	:4198	WORD[7FF8]	
	:4199		
	:4200		
U 00A8, 0080,00	:4201	WORD[8000]	:LOCATION 1B
U 00A9, 007F,FF	:4202	WORD[7FFF]	
	:4203		
U 00AA, 0000,00	:4204	WORD[0000]	:LOCATION 1C
U 00AB, 0000,00	:4205	WORD[0000]	
	:4206		
U 00AC, 0000,00	:4207	WORD[0000]	:LOCATION 1D
U 00AD, 0000,00	:4208	WORD[0000]	
	:4209		
U 00AE, 0005,00	:4210	WORD[0500]	:LOCATION 1E
U 00AF, 0000,80	:4211	WORD[0080]	
	:4212		
U 00B0, 0005,00	:4213	WORD[0500]	:LOCATION 1F
U 00B1, 0000,00	:4214	WORD[0000]	
	:4215		
U 00B2, 0000,01	:4216	WORD[0001]	:LOCATION 20
U 00B3, 0000,00	:4217	WORD[0000]	
	:4218		
U 00B4, 0000,02	:4219	WORD[0002]	:LOCATION 21
U 00B5, 0000,00	:4220	WORD[0000]	
	:4221		
U 00B6, 0000,04	:4222	WORD[0004]	:LOCATION 22
U 00B7, 0000,00	:4223	WORD[0000]	
	:4224		
U 00B8, 0000,08	:4225	WORD[0008]	:LOCATION 23
U 00B9, 0000,00	:4226	WORD[0000]	
	:4227		
U 00BA, 0000,10	:4228	WORD[0010]	:LOCATION 24
U 00BB, 0000,00	:4229	WORD[0000]	
	:4230		
U 00BC, 0000,20	:4231	WORD[0020]	:LOCATION 25
U 00BD, 0000,00	:4232	WORD[0000]	
	:4233		
U 00BE, 0000,40	:4234	WORD[0040]	:LOCATION 26
U 00BF, 0000,00	:4235	WORD[0000]	
	:4236		
U 00C0, 0000,80	:4237	WORD[0080]	:LOCATION 27
U 00C1, 0000,00	:4238	WORD[0000]	
	:4239		
U 00C2, 0001,00	:4240	WORD[0100]	:LOCATION 28
U 00C3, 0000,00	:4241	WORD[0000]	
	:4242		
U 00C4, 0002,00	:4243	WORD[0200]	:LOCATION 29
U 00C5, 0000,00	:4244	WORD[0000]	
	:4245		
U 00C6, 0004,00	:4246	WORD[0400]	:LOCATION 2A
U 00C7, 0000,00	:4247	WORD[0000]	
	:4248		
U 00C8, 0008,00	:4249	WORD[0800]	:LOCATION 2B
U 00C9, 0000,00	:4250	WORD[0000]	
	:4251		

U 00CA, 0010,00	:4252	WORD[1000]	:LOCATION 2C
U 00CB, 0000,00	:4253	WORD[0000]	
	:4254		
U 00CC, 0020,00	:4255	WORD[2000]	:LOCATION 2D
U 00CD, 0000,00	:4256	WORD[0000]	
	:4257		
U 00CE, 0040,00	:4258	WORD[4000]	:LOCATION 2E
U 00CF, 0000,00	:4259	WORD[0000]	
	:4260		
U 00D0, 0080,00	:4261	WORD[8000]	:LOCATION 2F
U 00D1, 0000,00	:4262	WORD[0000]	
	:4263		
U 00D2, 0000,00	:4264	WORD[0000]	:LOCATION 30
U 00D3, 0000,01	:4265	WORD[0001]	
	:4266		
U 00D4, 0000,00	:4267	WORD[0000]	:LOCATION 31
U 00D5, 0000,02	:4268	WORD[0002]	
	:4269		
U 00D6, 0000,00	:4270	WORD[0000]	:LOCATION 32
U 00D7, 0000,04	:4271	WORD[0004]	
	:4272		
U 00D8, 0000,00	:4273	WORD[0000]	:LOCATION 33
U 00D9, 0000,08	:4274	WORD[0008]	
	:4275		
U 00DA, 0000,00	:4276	WORD[0000]	:LOCATION 34
U 00DB, 0000,10	:4277	WORD[0010]	
	:4278		
U 00DC, 0000,00	:4279	WORD[0000]	:LOCATION 35
U 00DD, 0000,20	:4280	WORD[0020]	
	:4281		
U 00DE, 0000,00	:4282	WORD[0000]	:LOCATION 36
U 00DF, 0000,40	:4283	WORD[0040]	
	:4284		
U 00E0, 0000,00	:4285	WORD[0000]	:LOCATION 37
U 00E1, 0000,80	:4286	WORD[0080]	
	:4287		
U 00E2, 0000,00	:4288	WORD[0000]	:LOCATION 38
U 00E3, 0001,00	:4289	WORD[0100]	
	:4290		
U 00E4, 0000,00	:4291	WORD[0000]	:LOCATION 39
U 00E5, 0002,00	:4292	WORD[0200]	
	:4293		
U 00E6, 0000,00	:4294	WORD[0000]	:LOCATION 3A
U 00E7, 0004,00	:4295	WORD[0400]	
	:4296		
U 00E8, 0000,00	:4297	WORD[0000]	:LOCATION 3B
U 00E9, 0008,00	:4298	WORD[0800]	
	:4299		
U 00EA, 0000,00	:4300	WORD[0000]	:LOCATION 3C
U 00EB, 0010,00	:4301	WORD[1000]	
	:4302		
U 00EC, 0000,00	:4303	WORD[0000]	:LOCATION 3D
U 00ED, 0020,00	:4304	WORD[2000]	
	:4305		
U 00EE, 0000,00	:4306	WORD[0000]	:LOCATION 3E

ZZ-ENKCA-1.0	:	ENKCA.MIC							
: ENKCA.MCR			MICRO2 1L(03)	LS DATA SECTION	3-MAR-82 09:34:33	N 8	Fiche 1 Frame N8	Sequence 104	Page 98
: ENKCA.MIC			LS DATA SECTION				ENKCA Micro-diagnostic for DAP module	Rev 01.00	

U 00EF, 0040,00	:4307	WORD[4000]	
	:4308		
U 00F0, 0000,00	:4309	WORD[0000]	:LOCATION 3F
U 00F1, 0080,00	:4310	WORD[8000]	
	:4311		
U 00F2, 0000,00	:4312	WORD[0000]	:LOCATION 40
U 00F3, 0000,00	:4313	WORD[0000]	
	:4314		
U 00F4, 0000,00	:4315	WORD[0000]	:LOCATION 41
U 00F5, 0000,00	:4316	WORD[0000]	
	:4317		
U 00F6, 0000,00	:4318	WORD[0000]	:LOCATION 42
U 00F7, 0000,00	:4319	WORD[0000]	
	:4320		
U 00F8, 0000,00	:4321	WORD[0000]	:LOCATION 43
U 00F9, 0000,00	:4322	WORD[0000]	
	:4323		
U 00FA, 0000,00	:4324	WORD[0000]	:LOCATION 44
U 00FB, 0000,00	:4325	WORD[0000]	
	:4326		
U 00FC, 0000,00	:4327	WORD[0000]	:LOCATION 45
U 00FD, 0000,00	:4328	WORD[0000]	
	:4329		
U 00FE, 0000,00	:4330	WORD[0000]	:LOCATION 46
U 00FF, 0000,00	:4331	WORD[0000]	
	:4332		
U 0100, 0000,00	:4333	WORD[0000]	:LOCATION 47
U 0101, 0000,00	:4334	WORD[0000]	
	:4335		
U 0102, 0000,00	:4336	WORD[0000]	:LOCATION 48
U 0103, 0000,00	:4337	WORD[0000]	
	:4338		
U 0104, 0000,00	:4339	WORD[0000]	:LOCATION 49
U 0105, 0000,00	:4340	WORD[0000]	
	:4341		
U 0106, 0000,00	:4342	WORD[0000]	:LOCATION 4A
U 0107, 0000,00	:4343	WORD[0000]	
	:4344		
U 0108, 0055,55	:4345	WORD[5555]	:LOCATION 4B
U 0109, 00AA,AA	:4346	WORD[0AAAA]	
	:4347		
U 010A, 00AA,AA	:4348	WORD[0AAAA]	:LOCATION 4C
U 010B, 00AA,AA	:4349	WORD[0AAAA]	
	:4350		
U 010C, 0055,55	:4351	WORD[5555]	:LOCATION 4D
U 010D, 0055,55	:4352	WORD[5555]	
	:4353		
U 010E, 0000,00	:4354	WORD[0000]	:LOCATION 4E
U 010F, 0000,00	:4355	WORD[0000]	
	:4356		
U 0110, 00FF,FF	:4357	WORD[0FFFF]	:LOCATION 4F
U 0111, 00FF,FF	:4358	WORD[0FFFF]	
	:4359		
U 0112, 0000,00	:4360	WORD[0000]	:LOCATION 50
U 0113, 0000,00	:4361	WORD[0000]	

: ENKCA.MIC

MICRO2 1L(03)
LS DATA SECTION

LS DATA SECTION

3-MAR-82 09:34:33

3-MAR-82

ENKCA Micro-diagnostic for DAP module

Fiche 1 Frame B9

Sequence 105

Rev 01.00

Page 99

U	Address	Value	Label
U 0114,	0000,00	:4362	
U 0115,	0000,00	:4363	WORD[0000]
		:4364	:LOCATION 51
		:4365	
U 0116,	0000,00	:4366	WORD[0000]
U 0117,	0000,00	:4367	:LOCATION 52
		:4368	
U 0118,	0000,00	:4369	WORD[0000]
U 0119,	0000,00	:4370	:LOCATION 53
		:4371	
U 011A,	0000,00	:4372	WORD[0000]
U 011B,	0000,00	:4373	:LOCATION 54
		:4374	
U 011C,	0000,00	:4375	WORD[0000]
U 011D,	0000,00	:4376	:LOCATION 55
		:4377	
U 011E,	0000,00	:4378	WORD[0000]
U 011F,	0000,00	:4379	:LOCATION 56
		:4380	
U 0120,	0000,00	:4381	WORD[0000]
U 0121,	0000,00	:4382	:LOCATION 57
		:4383	
U 0122,	0000,00	:4384	WORD[0000]
U 0123,	0000,00	:4385	:LOCATION 58
		:4386	
U 0124,	0000,00	:4387	WORD[0000]
U 0125,	0000,00	:4388	:LOCATION 59
		:4389	
U 0126,	0000,00	:4390	WORD[0000]
U 0127,	0000,00	:4391	:LOCATION 5A
		:4392	
U 0128,	0000,00	:4393	WORD[0000]
U 0129,	0000,00	:4394	:LOCATION 5B
		:4395	
U 012A,	0000,00	:4396	WORD[0000]
U 012B,	0000,00	:4397	:LOCATION 5C
		:4398	
U 012C,	0000,00	:4399	WORD[0000]
U 012D,	0000,00	:4400	:LOCATION 5D
		:4401	
U 012E,	0000,00	:4402	WORD[0000]
U 012F,	0000,00	:4403	:LOCATION 5E
		:4404	
U 0130,	0000,00	:4405	WORD[0000]
U 0131,	0000,00	:4406	:LOCATION 5F
		:4407	
U 0132,	0000,03	:4408	WORD[0003]
U 0133,	0000,00	:4409	:LOCATION 60
		:4410	
U 0134,	0000,12	:4411	WORD[0012]
U 0135,	0000,00	:4412	:LOCATION 61
		:4413	
U 0136,	0033,33	:4414	WORD[3333]
U 0137,	0033,33	:4415	:LOCATION 62
		:4416	

zz... u u u u u u u u u u

U 0138, 000F,0F	:4417	WORD[0F0F]	:LOCATION 63
U 0139, 000F,0F	:4418	WORD[0F0F]	
	:4419		
U 013A, 0000,FF	:4420	WORD[00FF]	:LOCATION 64
U 013B, 0000,FF	:4421	WORD[00FF]	
	:4422		
U 013C, 0001,62	:4423	WORD[0162]	:LOCATION 65
U 013D, 0000,00	:4424	WORD[0000]	
	:4425		
U 013E, 0000,00	:4426	WORD[0000]	:LOCATION 66
U 013F, 0000,00	:4427	WORD[0000]	
	:4428		
U 0140, 0000,00	:4429	WORD[0000]	:LOCATION 67
U 0141, 0000,00	:4430	WORD[0000]	
	:4431		
U 0142, 0000,00	:4432	WORD[0000]	:LOCATION 68
U 0143, 0000,00	:4433	WORD[0000]	
	:4434		
U 0144, 0000,00	:4435	WORD[0000]	:LOCATION 69
U 0145, 0000,00	:4436	WORD[0000]	
	:4437		
U 0146, 0000,00	:4438	WORD[0000]	:LOCATION 6A
U 0147, 0000,00	:4439	WORD[0000]	
	:4440		
U 0148, 0000,00	:4441	WORD[0000]	:LOCATION 6B
U 0149, 0000,00	:4442	WORD[0000]	
	:4443		
U 014A, 0000,00	:4444	WORD[0000]	:LOCATION 6C
U 014B, 0000,00	:4445	WORD[0000]	
	:4446		
U 014C, 0000,00	:4447	WORD[0000]	:LOCATION 6D
U 014D, 0000,00	:4448	WORD[0000]	
	:4449		
U 014E, 0000,00	:4450	WORD[0000]	:LOCATION 6E
U 014F, 0000,00	:4451	WORD[0000]	
	:4452		
U 0150, 0000,00	:4453	WORD[0000]	:LOCATION 6F
U 0151, 0000,00	:4454	WORD[0000]	
	:4455		
U 0152, 0000,00	:4456	WORD[0000]	:LOCATION 70
U 0153, 0000,00	:4457	WORD[0000]	
	:4458		
U 0154, 0000,00	:4459	WORD[0000]	:LOCATION 71
U 0155, 0000,00	:4460	WORD[0000]	
	:4461		
U 0156, 0000,00	:4462	WORD[0000]	:LOCATION 72
U 0157, 0000,00	:4463	WORD[0000]	
	:4464		
U 0158, 0000,00	:4465	WORD[0000]	:LOCATION 73
U 0159, 0000,00	:4466	WORD[0000]	
	:4467		
U 015A, 0000,00	:4468	WORD[0000]	:LOCATION 74
U 015B, 0000,00	:4469	WORD[0000]	
	:4470		
U 015C, 0000,00	:4471	WORD[0000]	:LOCATION 75

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

MICRO2 1L(03)
LS DATA SECTION

LS DATA SECTION
3-MAR-82 09:34:33

D 9
3-MAR-82

ENKCA Micro-diagnostic for DAP module

Fiche 1 Frame D9

Sequence 107
Rev 01.00

Page 101

```
U 015D, 0000,00 :4472      WORD[0000]
                  :4473
U 015E, 0000,00 :4474      WORD[0000]      ;LOCATION 76
U 015F, 0000,00 :4475      WORD[0000]
                  :4476
U 0160, 0000,00 :4477      WORD[0000]      ;LOCATION 77
U 0161, 0000,00 :4478      WORD[0000]
                  :4479
                  :4480
                  :4481
                  :4482
                  :4483
                  :4484
                  :4485
                  :4486
.TOC      "PROGRAM SPECIFIC DATA SECTION (LS 80-F7)"
          This section represents the second half of LS. It is only acceseble
          with a MOV instruction, or one that uses the MOVE microinstruction for-
          mat. This section will be loaded in a sepearte transfer.
TRANSFER DATA LS2:
U 0162, 0000,78 :4487      COUNT.LS[0078]      ;LONGWORD COUNT (NUMBER OF LONGWORDS TO TRANS-
                  :4488      ;FER TO LS = 120 DECIMAL) DESTINATION IS LS
U 0163, 0000,80 :4489      START.ADR[0080]      ;DESTINATION START ADDRESS (START AT LS 80(H))
                  :4490
U 0164, 0000,3C :4491      WORD[003C]      ;LOCATION 80
U 0165, 0000,00 :4492      WORD[0000]
                  :4493
U 0166, 0000,43 :4494      WORD[0043]      ;LOCATION 81
U 0167, 0000,00 :4495      WORD[0000]
                  :4496
U 0168, 0000,64 :4497      WORD[0064]      ;LOCATION 82
U 0169, 0000,00 :4498      WORD[0000]
                  :4499
U 016A, 0000,25 :4500      WORD[0025]      ;LOCATION 83
U 016B, 0000,00 :4501      WORD[0000]
                  :4502
U 016C, 0000,26 :4503      WORD[0026]      ;LOCATION 84
U 016D, 0000,00 :4504      WORD[0000]
                  :4505
U 016E, 0000,20 :4506      WORD[0020]      ;LOCATION 85
U 016F, 0000,00 :4507      WORD[0000]
                  :4508
U 0170, 0000,54 :4509      WORD[0054]      ;LOCATION 86
U 0171, 0000,00 :4510      WORD[0000]
                  :4511
U 0172, 0000,7D :4512      WORD[007D]      ;LOCATION 87
U 0173, 0000,00 :4513      WORD[0000]
                  :4514
U 0174, 0000,00 :4515      WORD[0000]      ;LOCATION 88
U 0175, 0000,00 :4516      WORD[0000]
                  :4517
U 0176, 0000,00 :4518      WORD[0000]      ;LOCATION 89
U 0177, 0000,00 :4519      WORD[0000]
                  :4520
U 0178, 0000,00 :4521      WORD[0000]      ;LOCATION 8A
U 0179, 0000,00 :4522      WORD[0000]
                  :4523
U 017A, 0000,00 :4524      WORD[0000]      ;LOCATION 8B
U 017B, 0000,00 :4525      WORD[0000]
                  :4526
```


U 017C, 0000,00	:4527	WORD[0000]	:LOCATION 8C
U 017D, 0000,00	:4528	WORD[0000]	
	:4529		
U 017E, 0000,00	:4530	WORD[0000]	:LOCATION 8D
U 017F, 0000,00	:4531	WORD[0000]	
	:4532		
U 0180, 0000,00	:4533	WORD[0000]	:LOCATION 8E
U 0181, 0000,00	:4534	WORD[0000]	
	:4535		
U 0182, 0000,00	:4536	WORD[0000]	:LOCATION 8F
U 0183, 0000,00	:4537	WORD[0000]	
	:4538		
U 0184, 0000,00	:4539	WORD[0000]	:LOCATION 90
U 0185, 0000,00	:4540	WORD[0000]	
	:4541		
U 0186, 0055,55	:4542	WORD[5555]	:LOCATION 91
U 0187, 00AA,AA	:4543	WORD[0AAAA]	
	:4544		
U 0188, 0000,00	:4545	WORD[0000]	:LOCATION 92
U 0189, 0001,F0	:4546	WORD[01F0]	
	:4547		
U 018A, 0000,00	:4548	WORD[0000]	:LOCATION 93
U 018B, 0000,00	:4549	WORD[0000]	
	:4550		
U 018C, 0000,00	:4551	WORD[0000]	:LOCATION 94
U 018D, 0000,00	:4552	WORD[0000]	
	:4553		
U 018E, 00FF,FC	:4554	WORD[0FFFFC]	:LOCATION 95
U 018F, 00FF,FF	:4555	WORD[0FFFFF]	
	:4556		
U 0190, 0000,0F	:4557	WORD[000F]	:LOCATION 96
U 0191, 0000,00	:4558	WORD[0000]	
	:4559		
U 0192, 0000,F0	:4560	WORD[00F0]	:LOCATION 97
U 0193, 0000,00	:4561	WORD[0000]	
	:4562		
U 0194, 000F,00	:4563	WORD[0F00]	:LOCATION 98
U 0195, 0000,00	:4564	WORD[0000]	
	:4565		
U 0196, 00F0,00	:4566	WORD[0F000]	:LOCATION 99
U 0197, 0000,00	:4567	WORD[0000]	
	:4568		
U 0198, 0000,00	:4569	WORD[0000]	:LOCATION 9A
U 0199, 0000,03	:4570	WORD[0003]	
	:4571		
U 019A, 0080,00	:4572	WORD[8000]	:LOCATION 9B
U 019B, 007F,FF	:4573	WORD[7FFF]	
	:4574		
U 019C, 0000,00	:4575	WORD[0000]	:LOCATION 9C
U 019D, 0000,54	:4576	WORD[0054]	
	:4577		
U 019E, 0000,00	:4578	WORD[0000]	:LOCATION 9D
U 019F, 0000,F0	:4579	WORD[00F0]	
	:4580		
U 01A0, 0000,00	:4581	WORD[0000]	:LOCATION 9E

U 01A1, 0000,FC	:4582	WORD[00FC]	
U 01A2, 003F,FF	:4583	WORD[3FFF]	:LOCATION 9F
U 01A3, 003F,00	:4584	WORD[3F00]	
U 01A4, 0002,54	:4585	WORD[0254]	:LOCATION 0A0
U 01A5, 0000,00	:4586	WORD[0000]	
U 01A6, 0000,00	:4587	WORD[0000]	:LOCATION 0A1
U 01A7, 0000,00	:4588	WORD[0000]	
U 01A8, 0000,00	:4589	WORD[0000]	:LOCATION 0A2
U 01A9, 0000,00	:4590	WORD[0000]	
U 01AA, 0000,00	:4591	WORD[0000]	:LOCATION 0A3
U 01AB, 0000,00	:4592	WORD[0000]	
U 01AC, 0000,00	:4593	WORD[0000]	:LOCATION 0A4
U 01AD, 0000,00	:4594	WORD[0000]	
U 01AE, 0000,00	:4595	WORD[0000]	:LOCATION 0A5
U 01AF, 0000,00	:4596	WORD[0000]	
U 01B0, 0000,00	:4597	WORD[0000]	:LOCATION 0A6
U 01B1, 0000,00	:4598	WORD[0000]	
U 01B2, 0000,00	:4599	WORD[0000]	:LOCATION 0A7
U 01B3, 0000,00	:4600	WORD[0000]	
U 01B4, 0000,00	:4601	WORD[0000]	:LOCATION 0A8
U 01B5, 0000,00	:4602	WORD[0000]	
U 01B6, 0000,00	:4603	WORD[0000]	:LOCATION 0A9
U 01B7, 0000,00	:4604	WORD[0000]	
U 01B8, 0000,00	:4605	WORD[0000]	:LOCATION 0AA
U 01B9, 0000,00	:4606	WORD[0000]	
U 01BA, 0000,00	:4607	WORD[0000]	:LOCATION 0AB
U 01BB, 0000,00	:4608	WORD[0000]	
U 01BC, 0000,00	:4609	WORD[0000]	:LOCATION 0AC
U 01BD, 0000,00	:4610	WORD[0000]	
U 01BE, 0000,00	:4611	WORD[0000]	:LOCATION 0AD
U 01BF, 0000,00	:4612	WORD[0000]	
U 01C0, 0000,00	:4613	WORD[0000]	:LOCATION 0AE
U 01C1, 0000,00	:4614	WORD[0000]	
U 01C2, 0000,00	:4615	WORD[0000]	:LOCATION 0AF
U 01C3, 0000,00	:4616	WORD[0000]	
U 01C4, 0000,00	:4617	WORD[0000]	:LOCATION 0B0
U 01C5, 0000,00	:4618	WORD[0000]	

U 01C6, 0000.00	:4637	WORD[0000]	:LOCATION 0B1
U 01C7, 0000.00	:4638	WORD[0000]	
	:4639		
	:4640		
U 01C8, 0000.00	:4641	WORD[0000]	:LOCATION 0B2
U 01C9, 0000.00	:4642	WORD[0000]	
	:4643		
U 01CA, 0000.00	:4644	WORD[0000]	:LOCATION 0B3
U 01CB, 0000.00	:4645	WORD[0000]	
	:4646		
U 01CC, 0000.00	:4647	WORD[0000]	:LOCATION 0B4
U 01CD, 0000.00	:4648	WORD[0000]	
	:4649		
U 01CE, 0000.00	:4650	WORD[0000]	:LOCATION 0B5
U 01CF, 0000.00	:4651	WORD[0000]	
	:4652		
U 01D0, 0000.00	:4653	WORD[0000]	:LOCATION 0B6
U 01D1, 0000.00	:4654	WORD[0000]	
	:4655		
U 01D2, 0000.00	:4656	WORD[0000]	:LOCATION 0B7
U 01D3, 0000.00	:4657	WORD[0000]	
	:4658		
U 01D4, 0000.00	:4659	WORD[0000]	:LOCATION 0B8
U 01D5, 0000.00	:4660	WORD[0000]	
	:4661		
U 01D6, 0000.00	:4662	WORD[0000]	:LOCATION 0B9
U 01D7, 0000.00	:4663	WORD[0000]	
	:4664		
U 01D8, 0000.00	:4665	WORD[0000]	:LOCATION 0BA
U 01D9, 0000.00	:4666	WORD[0000]	
	:4667		
U 01DA, 0000.00	:4668	WORD[0000]	:LOCATION 0BB
U 01DB, 0000.00	:4669	WORD[0000]	
	:4670		
U 01DC, 0000.00	:4671	WORD[0000]	:LOCATION 0BC
U 01DD, 0000.00	:4672	WORD[0000]	
	:4673		
U 01DE, 0000.00	:4674	WORD[0000]	:LOCATION 0BD
U 01DF, 0000.00	:4675	WORD[0000]	
	:4676		
U 01E0, 0000.00	:4677	WORD[0000]	:LOCATION 0BE
U 01E1, 0000.00	:4678	WORD[0000]	
	:4679		
U 01E2, 0000.00	:4680	WORD[0000]	:LOCATION 0BF
U 01E3, 0000.00	:4681	WORD[0000]	
	:4682		
U 01E4, 0000.00	:4683	WORD[0000]	:LOCATION 0C0
U 01E5, 0000.00	:4684	WORD[0000]	
	:4685		
U 01E6, 0000.00	:4686	WORD[0000]	:LOCATION 0C1
U 01E7, 0000.00	:4687	WORD[0000]	
	:4688		
U 01E8, 0000.00	:4689	WORD[0000]	:LOCATION 0C2
U 01E9, 0000.00	:4690	WORD[0000]	
	:4691		

U 01EA, 0000.00	:4692	WORD[0000]	:LOCATION 0C3
U 01EB, 0000.00	:4693	WORD[0000]	
	:4694		
U 01EC, 0000.00	:4695	WORD[0000]	:LOCATION 0C4
U 01ED, 0000.00	:4696	WORD[0000]	
	:4697		
U 01EE, 0000.00	:4698	WORD[0000]	:LOCATION 0C5
U 01EF, 0000.00	:4699	WORD[0000]	
	:4700		
U 01F0, 0000.00	:4701	WORD[0000]	:LOCATION 0C6
U 01F1, 0000.00	:4702	WORD[0000]	
	:4703		
U 01F2, 0000.00	:4704	WORD[0000]	:LOCATION 0C7
U 01F3, 0000.00	:4705	WORD[0000]	
	:4706		
U 01F4, 0000.00	:4707	WORD[0000]	:LOCATION 0C8
U 01F5, 0000.00	:4708	WORD[0000]	
	:4709		
U 01F6, 0000.00	:4710	WORD[0000]	:LOCATION 0C9
U 01F7, 0000.00	:4711	WORD[0000]	
	:4712		
U 01F8, 0000.00	:4713	WORD[0000]	:LOCATION 0CA
U 01F9, 0000.00	:4714	WORD[0000]	
	:4715		
U 01FA, 0000.00	:4716	WORD[0000]	:LOCATION 0CB
U 01FB, 0000.00	:4717	WORD[0000]	
	:4718		
U 01FC, 0000.00	:4719	WORD[0000]	:LOCATION 0CC
U 01FD, 0000.00	:4720	WORD[0000]	
	:4721		
U 01FE, 0000.00	:4722	WORD[0000]	:LOCATION 0CD
U 01FF, 0000.00	:4723	WORD[0000]	
	:4724		
U 0200, 0000.00	:4725	WORD[0000]	:LOCATION 0CE
U 0201, 0000.00	:4726	WORD[0000]	
	:4727		
U 0202, 0000.00	:4728	WORD[0000]	:LOCATION 0CF
U 0203, 0000.00	:4729	WORD[0000]	
	:4730		
U 0204, 0000.00	:4731	WORD[0000]	:LOCATION 0D0
U 0205, 0000.00	:4732	WORD[0000]	
	:4733		
U 0206, 0000.00	:4734	WORD[0000]	:LOCATION 0D1
U 0207, 0000.00	:4735	WORD[0000]	
	:4736		
U 0208, 0000.00	:4737	WORD[0000]	:LOCATION 0D2
U 0209, 0000.00	:4738	WORD[0000]	
	:4739		
U 020A, 0000.00	:4740	WORD[0000]	:LOCATION 0D3
U 020B, 0000.00	:4741	WORD[0000]	
	:4742		
U 020C, 0000.00	:4743	WORD[0000]	:LOCATION 0D4
U 020D, 0000.00	:4744	WORD[0000]	
	:4745		
U 020E, 0000.00	:4746	WORD[0000]	:LOCATION 0D5

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

MICRO2 1L(03) PROGRAM SPECIFIC DA 3-MAR-82
3-MAR-82 09:34:33
PROGRAM SPECIFIC DATA SECTION (LS 80-F7)

Fiche 1 Frame I9
ENKCA Micro-diagnostic for DAP module
Sequence 112
Rev 01.00
Page 106

U 020F, 0000.00	:4747	WORD[0000]	
	:4748		
U 0210, 0000.00	:4749	WORD[0000]	:LOCATION 0D6
U 0211, 0000.00	:4750	WORD[0000]	
	:4751		
U 0212, 0000.00	:4752	WORD[0000]	:LOCATION 0D7
U 0213, 0000.00	:4753	WORD[0000]	
	:4754		
U 0214, 0000.00	:4755	WORD[0000]	:LOCATION 0D8
U 0215, 0000.00	:4756	WORD[0000]	
	:4757		
U 0216, 0000.00	:4758	WORD[0000]	:LOCATION 0D9
U 0217, 0000.00	:4759	WORD[0000]	
	:4760		
U 0218, 0000.00	:4761	WORD[0000]	:LOCATION 0DA
U 0219, 0000.00	:4762	WORD[0000]	
	:4763		
U 021A, 0000.00	:4764	WORD[0000]	:LOCATION 0DB
U 021B, 0000.00	:4765	WORD[0000]	
	:4766		
U 021C, 0000.00	:4767	WORD[0000]	:LOCATION 0DC
U 021D, 0000.00	:4768	WORD[0000]	
	:4769		
U 021E, 0000.00	:4770	WORD[0000]	:LOCATION 0DD
U 021F, 0000.00	:4771	WORD[0000]	
	:4772		
U 0220, 0000.00	:4773	WORD[0000]	:LOCATION 0DE
U 0221, 0000.00	:4774	WORD[0000]	
	:4775		
U 0222, 0000.00	:4776	WORD[0000]	:LOCATION 0DF
U 0223, 0000.00	:4777	WORD[0000]	
	:4778		
U 0224, 0000.00	:4779	WORD[0000]	:LOCATION 0E0
U 0225, 0000.00	:4780	WORD[0000]	
	:4781		
U 0226, 0000.00	:4782	WORD[0000]	:LOCATION 0E1
U 0227, 0000.00	:4783	WORD[0000]	
	:4784		
U 0228, 0000.00	:4785	WORD[0000]	:LOCATION 0E2
U 0229, 0000.00	:4786	WORD[0000]	
	:4787		
U 022A, 0000.00	:4788	WORD[0000]	:LOCATION 0E3
U 022B, 0000.00	:4789	WORD[0000]	
	:4790		
U 022C, 0000.00	:4791	WORD[0000]	:LOCATION 0E4
U 022D, 0000.00	:4792	WORD[0000]	
	:4793		
U 022E, 0000.00	:4794	WORD[0000]	:LOCATION 0E5
U 022F, 0000.00	:4795	WORD[0000]	
	:4796		
U 0230, 0000.00	:4797	WORD[0000]	:LOCATION 0E6
U 0231, 0000.00	:4798	WORD[0000]	
	:4799		
U 0232, 0000.00	:4800	WORD[0000]	:LOCATION 0E7
U 0233, 0000.00	:4801	WORD[0000]	

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

PROGRAM SPECIFIC DATA SECTION (LS 80-F7)
MICRO2 1L(03) 3-MAR-82 09:34:33
Fiche 1 Frame J9
ENKCA Micro-diagnostic for DAP module
Sequence 113
Rev 01.00
Page 107

U 0234, 0000.00	:4802	WORD[0000]	:LOCATION 0E8
U 0235, 0000.00	:4803	WORD[0000]	
	:4804		
U 0236, 0000.00	:4805		
U 0237, 0000.00	:4806	WORD[0000]	:LOCATION 0E9
	:4807	WORD[0000]	
	:4808		
U 0238, 0000.00	:4809	WORD[0000]	:LOCATION 0EA
U 0239, 0000.00	:4810	WORD[0000]	
	:4811		
U 023A, 0000.00	:4812	WORD[0000]	:LOCATION 0EB
U 023B, 0000.00	:4813	WORD[0000]	
	:4814		
U 023C, 0000.00	:4815	WORD[0000]	:LOCATION 0EC
U 023D, 0000.00	:4816	WORD[0000]	
	:4817		
U 023E, 0000.00	:4818	WORD[0000]	:LOCATION 0ED
U 023F, 0000.00	:4819	WORD[0000]	
	:4820		
U 0240, 0000.00	:4821	WORD[0000]	:LOCATION 0EE
U 0241, 0000.00	:4822	WORD[0000]	
	:4823		
U 0242, 0000.00	:4824	WORD[0000]	:LOCATION 0EF
U 0243, 0000.00	:4825	WORD[0000]	
	:4826		
U 0244, 0000.00	:4827	WORD[0000]	:LOCATION 0F0
U 0245, 0000.00	:4828	WORD[0000]	
	:4829		
U 0246, 0000.00	:4830	WORD[0000]	:LOCATION 0F1
U 0247, 0000.00	:4831	WORD[0000]	
	:4832		
U 0248, 0000.00	:4833	WORD[0000]	:LOCATION 0F2
U 0249, 0000.00	:4834	WORD[0000]	
	:4835		
U 024A, 0000.00	:4836	WORD[0000]	:LOCATION 0F3
U 024B, 0000.00	:4837	WORD[0000]	
	:4838		
U 024C, 0000.00	:4839	WORD[0000]	:LOCATION 0F4
U 024D, 0000.00	:4840	WORD[0000]	
	:4841		
U 024E, 0000.00	:4842	WORD[0000]	:LOCATION 0F5
U 024F, 0000.00	:4843	WORD[0000]	
	:4844		
U 0250, 0000.00	:4845	WORD[0000]	:LOCATION 0F6
U 0251, 0000.00	:4846	WORD[0000]	
	:4847		
U 0252, 0000.00	:4848	WORD[0000]	:LOCATION 0F7
U 0253, 0000.00	:4849	WORD[0000]	

: IF ANY TRANSFER DATA IS ADDED, PUT BEFORE THIS
: LINE. LIMIT OF 962 ADDITIONAL WORDS (470
: LONGWORD TRANSFERS).

.REGION/600,6FF


```

:4856 .PAGE 'WCS SUBROUTINES FOR CONSOLE USE'
:4857 .TOC  " GENERATE TRANSFER DATA"
:4858
:4859 This routine is used by the diagnostic monitor to load WR 0 with a
:4860 data pattern from the console. The monitor will set CONSOLE ATTN if
:4861 a 1 is to be generated, or clear CONSOLE ATTN if a 0 is to be gen-
:4862 erated. It will then single step this routine to shift the data bit
:4863 into WR 0. This routine loads a 32 bit value much more quickly than
:4864 shifting each instruction into the CSR from the monitor, and is used
:4865 mainly to set up data to load in LS for constants.
:4866
:4867
:4868 ***WARNING***
:4869
:4870 This routine must start at 600 and end at 605. DO NOT move this
:4871 routine to any other area!!
:4872
:4873
:4874
:4875 GEN.XFER.DATA:
:4876 CLP WR[0] ;CLEAR A WORKING REG
:4877 COM WR[0] ;NOW WR 0 IS ALL ONES
:4878
:4879 LOOP.XFER:
:4880 SKIP.IF[CONSOLE.ATTN] ;SKIP NEXT INSTRUCTION IS CONSOLE ATTN
:4881 ; IS SET (GENERATING A 1)
:4882 ASHL WR[0], ;SHIFT A 0 INTO BIT 0 OF WR 0
:4883 SKIP ;AND SKIP ROL
:4884 ROL WR[0] ;SHIFT A 1 INTO BIT 0 OF WR 0
:4885 JMP [LOOP.XFER] ;REPEAT
:4886
:4887
:4888 DIAGNOSTIC VERSION NUMBER IS STORED HERE
:4889
:4890
:4891 ***WARNING***
:4892
:4893 These words must be at location 606 and 607. DO NOT move this word
:4894 to any other area!!
:4895
:4896
:4897
:4898 U 0606, 0001,00 WORD[0100] ;VERSION 01.00
:4899 U 0607, 0043,41 ASCII [CA] ;LAST 2 LETTERS OF FILE NAME [ENKCA]
:4900
:4901

```

```
:4902 .PAGE  " DEPOSIT MCT CSR ROUTINE"
:4903 .+++
:4904
:4905 .FUNCTIONAL DESCRIPTION:
:4906
:4907 .The deposit to CSR routine is used to write the memory controller's
:4908 .control and status register. The memory controller has three CSRs
:4909 .named CSR 0, CSR 1, and CSR 2.
:4910 .CSR 0 contains checkbits or syndrome bits returned from the ECC logic
:4911 .after certain diagnostic routines. It is NOT writable directly with
:4912 .this routine.
:4913 .CSR 1 contains error bits set if an error occurs on a memory access.
:4914 .It does contain five maintenance bits (bits 29-25) which are writable.
:4915 .There are also seven checkbits (bits 6-0) which are writable, but not
:4916 .readable. These bits are used to write checkbits into the ECC chips.
:4917 .CSR 2 contains error bits set if a UNIBUS error occurs on a UNIBUS
:4918 .access. These are read only bits and are NOT writable by this routine.
:4919 .Therefore, CSR 1 is the only writable CSR, and only bits 29-25 can be
:4920 .both written and read back. This routine will thus force CSR 1 on a
:4921 .deposit to CSR.
:4922 .This routine will write LS 5 with data to access CSR 1, then write
:4923 .the data residing in LS 6 into the CSR. It will then issue CPU ATTN
:4924 .to inform the console that it has completed the function.
:4925 .NOTE: The error bits of CSR 1 are cleared on any write to CSR 1.
:4926 .These are bits 31, 30 and 23-14. Bits 13-0 and bit 24 are unused.
:4927
:4928 .CALLING SEQUENCE:
:4929
:4930 .The console loads the address of a pointer to this routine (a JMP in-
:4931 .struction at WCS location 50) into the UPC and starts the CPU.
:4932
:4933 .INPUT PARAMETERS:
:4934
:4935 .LS location 6 must have been written by the console with the desired
:4936 .data to be deposited in CSR 1 previous to executing this routine.
:4937
:4938 .IMPLICIT INPUTS:
:4939
:4940 .None
:4941
:4942 .OUTPUT PARAMATERS:
:4943
:4944 .CSR 1 bits 29-25 and 6-0 are written with data from LS 6.
:4945
:4946 .IMPLICIT OUTPUTS:
:4947
:4948 .None
:4949
:4950 .SIDE EFFECTS:
:4951
:4952 .WR 0 will be modified by this routine.
:4953 .LS 5 will be modified by this routine.
:4954 .CSR 1 bits 31, 30 and 23-14 are cleared.
:4955
:4956 .---
```

	:4957		
	:4958	DEPOSIT.CSR:	
	:4959		
U 0608, 3644,15	:4960	MOV LS[CSR1] TO WR[0]	:SET BIT 2 IN WR 0 AS CSR NUMBER TO
	:4961		:WRITE (BITS 2 AND 3 ARE CSR NUMBER)
U 0609, BE0A,15	:4962	MOV WR[0] TO LS[T5.SUB]	:PUT CSR NUMBER IN LS LOCATION 5 (CSR
	:4963		:1)
U 060A, 1DOB,F5	:4964	MEM.REQ[WRITE.CSR] ADRS[T5.SUB]	:ISSUE MEMORY REQUEST TO
	:4965		:ACCESS CSR 1
U 060B, B20C,15	:4966	WRITE.MEM LS[T6.SUB]	:SEND DATA IN LS LOCATION 6 TO CSR 1
U 060C, 8868,74	:4967	JMP [WAIT.SUB]	:JUMP TO SET ATTN AND WAIT

:4968 .PAGE " EXAMINE MCT CSR ROUTINE"

:4969 :+++

:4970

:4971

:4972

:4973

:4974

:4975

:4976

:4977

:4978

:4979

:4980

:4981

:4982

:4983

:4984

:4985

:4986

:4987

:4988

:4989

:4990

:4991

:4992

:4993

:4994

:4995

:4996

:4997

:4998

:4999

:5000

:5001

:5002

:5003

:5004

:5005

:5006

:5007

:5008

:5009

:5010

:5011

:5012

:5013

:5014

:5015

:5016

:5017

:5018

:5019

:5020

:5021

:5022

FUNCTIONAL DESCRIPTION:

The examine CSR routine reads the memory controller's control and status register. There are 3 CSRs labeled CSR 0, CSR 1 and CSR 2.

CSR 0 contains checkbits or syndrome bits from the ECC logic. These are contained in bits 6-0 of the CSR. Other bits are UNPREDICTABLE. Bits 6-0 are only written by special diagnostic routines (they are not writable by the DEPOSIT CSR command).

CSR 1 contains error bits and maintenance bits. The error bits are 31, 30, and 23-14. Maintenance bits are 29-25. The other bits are UNPREDICTABLE. The error bits are written during memory functions only and are not writable via the DEPOSIT CSR command. The maintenance bits are writable. Note that a DEPOSIT CSR command will write the maintenance bits and clear all error bits.

CSR 2 contains three bits (bits 16-14) which are readable. These are set when a UNIBUS error occurs on a UNIBUS access. They are not directly writable with the DEPOSIT CSR command.

The console loads LS 5 with the CSR number (0, 1, or 2) to be read. This routine rotates LS 5 two places left to position it correctly. It then executes a read CSR and places the data in LS 6 for the console to read.

CALLING SEQUENCE:

The console loads the address of a pointer to this routine (a JMP instruction at WCS location 51) into the UPC and starts the CPU.

INPUT PARAMETERS:

LS 5 contains the CSR number to be read.

IMPLICIT INPUTS:

None

OUTPUT PARAMETERS:

LS 6 - Data read from CSR selected.

IMPLICIT OUTPUTS:

None

SIDE EFFECTS:

LS 5 is rotated 2 places left.

EXAMINE.CSR:

U 060D, 360A,15

MOV LS[T5.SUB] TO WR[0]

;GET CSR NUMBER FROM LS 5

B 10
 3-MAR-82
 09:34:33
 FICHE 1 Frame B10
 ENKCA Micro-diagnostic for DAP module
 Sequence 118
 Rev 01.00
 Page 112

```

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC
MICRO2 1L(03) EXAMINE MCT CSR ROUTINE
EXAMINE MCT CSR ROUTINE

U 060E, A2D0,15 :5023      SHL2 WR[0]                ;SHIFT NUMBER 2 PLACES LEFT TO GET
:5024                ;CSR NUMBER IN BITS 2 AND 3
U 060F, BE0A,15 :5025      MOV WR[0] TO LS[T5.SUB]        ;PUT SHIFTED NUMBER IN LS 5
:5026
U 0610, 9D0B,75 :5027      MEM.REQ[READ.CSR] ADRS[T5.SUB] DT[LONG] ;ISSUE MEMORY REQUEST TO
:5028                ;ACCESS CSR SPECIFIED
U 0611, 3022,15 :5029      MOV MEM.DATA TO WR[0]          ;READ CSR SELECTED
:5030
:5031
U 0612, B60A,95 :5032      CHECK.CSR2:
:5033      MOV LS[T5.SUB] TO WR[1]                ;GET SHIFTED CSR NUMBER
:5034      CMP LS[#8] WITH WR[1],                ;SEE IF CSR 2 WAS SELECTED
:5035      DT(LONG)&SET.ALU.CC                    ;SET UP CONDITION CODES
:5036      JMP.IF[NEQ] TO [CHECK.CSR1]           ;JUMP IF CSR 2 WAS NOT SELECTED
:5037      BIC LS[CSR2.MASK] TO WR[0]            ;CLEAR UNUSED (UNPREDICTABLE) BITS IN
:5038                ;CSR 2.
U 0616, 4E44,B5 :5039      CHECK.CSR1:
:5040      CMP LS[#4] WITH WR[1],                ;SEE IF CSR 1 WAS SELECTED
:5041      DT(LONG)&SET.ALU.CC                    ;SET UP CONDITION CODES
:5042      JMP.IF[NEQ] TO [CHECK.CSR0]           ;JUMP IF CSR 1 WAS NOT SELECTED
:5043      BIC LS[CSR1.MASK] TO WR[0]            ;CLEAR UNUSED (UNPREDICTABLE) BITS IN
:5044                ;CSR 1.
U 0619, 4E9C,B5 :5045      CHECK.CSR0:
:5046      CMP LS[#0] WITH WR[1],                ;SEE IF CSR 0 WAS SELECTED
:5047      DT(LONG)&SET.ALU.CC                    ;SET UP CONDITION CODES
:5048      JMP.IF[NEQ] TO [LOAD.LS6]            ;JUMP IF CSR 0 WAS NOT SELECTED
:5049      BIC LS[CSR0.MASK] TO WR[0]            ;CLEAR UNUSED (UNPREDICTABLE) BITS IN
:5050                ;CSR 0.
U 061C, C54E,15 :5050      BIC LS[BIT7] TO WR[0]        ;BIT 7 IS ALSO UNUSED
:5051
:5052
U 061D, BE0C,15 :5053      LOAD.LS6:
:5054      MOV WR[0] TO LS[T6.SUB]                ;LOAD CSR DATA INTO LS 6 FOR PRINTOUT
:5055      JMP [WAIT.SUB]                        ;ISSUE CPU ATTN AND WAIT FOR RESPONSE
  
```

```
5055 .page  " DEPOSIT MCT TRANSLATION BUFFER ROUTINE"
5056 .+++
5057
5058 .FUNCTIONAL DESCRIPTION:
5059
5060 .    This routine allows a write to the translation buffer portion of the
5061 .    memory controller. The buffer area contains 128 decimal locations
5062 .    addressed from 0-7F(H). The remainder of the TB RAM is either unused
5063 .    or contains the UNIBUS MAP. The DEPOSIT UB command is used to write
5064 .    the UNIBUS map portion of the TB RAMs. The address specified with the
5065 .    DEPOSIT command will be the actual RAM address accessed. This routine
5066 .    will do any shifting necessary on the address specified to position it
5067 .    correctly. The address specified has been loaded by the console in LS
5068 .    location 5 prior to executing this routine. The address must be in HEX
5069 .    format.
5070 .    The data specified has been placed in LS 6 by the console prior to
5071 .    executing this routine. Bits 07-01 will be the Valid, Prot, Modify,
5072 .    and Byte Offset bits (there are four Prot bits). Bits 22-08 will
5073 .    contain the PA bits from PA 23 through PA 09 respectively. Bits 31
5074 .    through 23 and bit 0 are unused. This routine will do any shifting nec-
5075 .    essary to write the bits in TB as described. The data specified must
5076 .    be in HEX format.
5077
5078 .CALLING SEQUENCE:
5079
5080 .    The console loads the address of a pointer to this routine (a JMP in-
5081 .    struction at WCS location 52) into the UPC and starts the CPU.
5082
5083 .INPUT PARAMETERS:
5084
5085 .    LS 5 - TB address to be written (range 0-7F)
5086 .    LS 6 - TB data to write (22 bits, from 1-22). Bits 0 and 23-31 are
5087 .    ignored.
5088
5089 .IMPLICIT INPUTS:
5090
5091 .    None
5092
5093 .OUTPUT PARAMATERS:
5094
5095 .    TB will be written with specified data at specified address
5096
5097 .IMPLICIT OUTPUTS:
5098
5099 .    None
5100
5101 .SIDE EFFECTS:
5102
5103 .    WR 0 will be modified
5104 .    WR 1 will be modified
5105
5106 .---
5107
5108 .DEPOSIT.TB:
5109 .    JSR [SHIFT.TB.ADR]          ;GO TO SUBROUTINE TO POSITION TB AD-
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) DEPOSIT MCT TRANSLA 3-MAR-82
3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
DEPOSIT MCT TRANSLATION BUFFER ROUTINE

D 10

Fiche 1 Frame D10

Sequence 120

Rev 01.00

Page 114

```
:5110
U 0620, 360C,15 :5111      MOV LS[T6.SUB] TO WR[0]      : DRESS CORRECTLY
U 0621, 4526,15 :5112      BIC LS[7FF] TO WR[0]      : GET DATA FROM LS 6
:5113      :5113      : CLEAR LOW BYTE OF DATA SPECIFIED.
:5114      :5114      : LOW BYTE WILL BE PUT IN BITS 24-31
:5115      :5115      : LATER
U 0622, 8862,AC :5116      JSR [SHIFT.LOOP]      : GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
:5117      :5117      : PLACES RIGHT
U 0623, E40C,15 :5118      SWAP LS[T6.SUB] WITH WR[0]      : SWAP ORIGINAL DATA SPECIFIED WITH MOD-
:5119      :5119      : IFIED DATA. NOW LS 6 CONTAINS BITS
:5120      :5120      : 8-22 IN BITS 0-14 AS REQUIRED. WR 0
:5121      :5121      : CONTAINS ORIGINAL DATA SPECIFIED
U 0624, 452C,15 :5122      BIC LS[7FFFFFF0] TO WR[0]      : CLEAR ALL BUT LOW BYTE OF ORIGINAL
:5123      :5123      : DATA SPECIFIED
U 0625, 8862,AC :5124      JSR [SHIFT.LOOP]      : GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
:5125      :5125      : PLACES RIGHT. ON RETURN, WR 0 WILL
:5126      :5126      : HAVE BITS 0-7 IN BITS 24-31, OTHER
:5127      :5127      : BITS ARE CLEAR
U 0626, EDOC,15 :5128      BIS WR[0] TO LS[T6.SUB]      : NOW LS 6 CONTAINS DATA TO WRITE IN
:5129      :5129      : TB IN CORRECT FORMAT I.E. PA BITS
:5130      :5130      : IN BITS 00-14 AND BYTE OFFSET, MODIFY
:5131      :5131      : PROT A THROUGH PROT D, AND VALID IN
:5132      :5132      : BITS 25-31.
U 0627, 980A,F5 :5133      MEM.REQ[WRITE.TB] ADRS[T5.SUB] DT[LONG] : ISSUE MEMORY REQUEST TO WRITE
:5134      :5134      : THE TB
U 0628, B20C,15 :5135      WRITE.MEM LS[T6.SUB]      : WRITE DATA FROM LS 6 IN TB
U 0629, 8868,74 :5136      JMP [WAIT.SUB]      : JUMP TO SET ATTN AND WAIT
:5137      :5137
:5138      :5138
:5139      :5139      : SUBROUTINE TO SHIFT WR 0 TO THE RIGHT 8 PLACES
:5140      :5140
:5141      :5141      : SHIFT.LOOP:
U 062A, 3646,95 :5142      MOV LS[8] TO WR[1]      : COUNTER IN WR 1 TO SHIFT 8 PLACES
:5143      :5143      : SHIFT.LOOP.1:
U 062B, 2340,15 :5144      ROR WR[0]      : SHIFT WR 0 ONE PLACE RIGHT
:5145      :5145      : DECREMENT COUNTER
U 062C, A102,B5 :5146      DT[LONG]&SET.ALU.CC      : AND SET CONDITION CODES
U 062D, 8862,B1 :5147      JMP.IF[NEQ] TO [SHIFT.LOOP.1] : REPEAT UNTIL 8 SHIFTS HAVE OCCURRED
U 062E, 5B00,14 :5148      RETURN      : THEN RETURN
:5149      :5149
:5150      :5150      : SUBROUTINE TO POSITION TB ADDRESS IN BITS 9-14 AND BIT 31 (TB ADDRESS
:5151      :5151      : BIT 0 MUST BE IN BIT 31).
:5152      :5152
:5153      :5153      : SHIFT.TB.ADR:
U 062F, 360A,15 :5154      MOV LS[T5.SUB] TO WR[0]      : GET TB ADDRESS FROM LS 5
U 0630, 452C,15 :5155      BIC LS[7FFFFFF0] TO WR[0]      : CLEAR ALL BUT LOW BYTE (8 BIT ADDRESS)
U 0631, A2D0,15 :5156      SHL2 WR[0]      : SHIFT 2 PLACES LEFT TO POSITION ADDRESS
U 0632, A2D0,15 :5157      SHL2 WR[0]      : SHIFT 2 PLACES LEFT TO POSITION ADDRESS
U 0633, A2D0,15 :5158      SHL2 WR[0]      : SHIFT 2 PLACES LEFT TO POSITION ADDRESS
U 0634, A2D0,15 :5159      SHL2 WR[0]      : SHIFT 2 PLACES LEFT TO POSITION ADDRESS
U 0635, 23D0,15 :5160      ASHL WR[0]      : SHIFT ONE MORE PLACE
:5161      :5161      : NOW BITS 0-6 IN BITS 9-15
:5162      :5162      : SEE IF BIT 15 (MSB) IS SET OR CLEAR.
U 0636, 595E,35 :5163      BIT LS[BIT15] WITH WR[0],      : MSB MUST GO IN BIT 31 IN ORDER TO AD-
:5164      :5164      : DT[LONG]&SET.ALU.CC      : DRESS TB CORRECTLY
```

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

DEPOSIT MCT TRANSLA 3-MAR-82 E 10
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
DEPOSIT MCT TRANSLATION BUFFER ROUTINE

Fiche 1 Frame E10

Sequence 121

Rev 01.00 Page 115

U 0637, 5B00,09 :5165
U 0638, 477E,15 :5166
 :5167
 :5168
 :5169
U 0639, 3E0A,14 :5170

SKIP.IF[EQL]
BIS LS[BIT31] TO WR[0]
MOV WR[0] TO LS[T5.SUB],
RETURN

:SKIP NEXT INSTRUCTION IF MSB IS 0
:ELSE SET BIT 31 FOR MSB
:NOW TB ADDRESS IS IN CORRECT FORM IN
:LS LOCATION 5
: THEN RETURN TO MAIN CODE

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) EXAMINE TRANSLATION 3-MAR-82
EXAMINE TRANSLATION BUFFER ROUTINE

F 10
3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module

Fiche 1 Frame F10

Sequence 122
Rev 01.00 Page 116

```
:5171 .PAGE " EXAMINE TRANSLATION BUFFER ROUTINE"
:5172 .+++
:5173
:5174 .FUNCTIONAL DESCRIPTION:
:5175
:5176 .This routine will read the translation buffer of the memory controller.
:5177 .The TB will be returned with the byte offset, modify, prot A through
:5178 .prot D, and valid in bits 01-07 respectively. PA bits 09-23 are re-
:5179 .turned in bits 08-22 respectively. Bits 23-31 and bit 0 are not used
:5180 .and so will be cleared before printing the result. The result will be
:5181 .put in LS 6 for the console to print.
:5182 .The routine will issue a memory request with memory function=READ.TB
:5183 .and data type=LONGWORD. The console will load LS 5 with the address
:5184 .specified before executing this routine. The routine will shift the
:5185 .address around as required to address the TB location specified. The
:5186 .address specified must be in HEX format.
:5187 .The TB consists of 128 decimal locations (addresses from 0-7F). The
:5188 .remaining locations in the TB RAM are either unused or used by the
:5189 .UNIBUS map. UNIBUS map addresses are read via the EXAMINE UB routine.
:5190
:5191 .CALLING SEQUENCE:
:5192
:5193 .The console loads the address of a pointer to this routine (a JMP in-
:5194 .struction at WCS location 53) into the UPC and starts the CPU.
:5195
:5196 .INPUT PARAMETERS:
:5197
:5198 .LS 5 - TB address (0-7F) in hex format
:5199
:5200 .IMPLICIT INPUTS:
:5201
:5202 .None
:5203
:5204 .OUTPUT PARAMATERS:
:5205
:5206 .LS 6 - TB data from address specified
:5207
:5208 .IMPLICIT OUTPUTS:
:5209
:5210 .None
:5211
:5212 .SIDE EFFECTS:
:5213
:5214 .LS 5 is modified
:5215 .WR 0 is modified
:5216
:5217 .---
:5218
:5219 .EXAMINE.TB:
:5220 .JSR [SHIFT.TB.ADR] ;SHIFT ADDRESS SPECIFIED FROM LS 5 INTO
:5221 . ; CORRECT POSITION AND REPLACE IN LS 5
:5222 .MEM.REQ[READ.TB] ADRS[T5.SUB] DT[LONG] ;ISSUE MEMORY REQUEST TO READ
:5223 . ; THE TB AT ADDRESS CONTAINED IN LS 5
:5224 .MOV MEM.DATA TO WR[0] ;PLACE RESULT IN WR 0
:5225
```

U 063A, 8862,FC

U 063B, 9C0B,F5

U 063C, 3022,15

ZZ-ENKCA-1.0	:	ENKCA.MIC							
:	:	ENKCA.MCR							
:	:	ENKCA.MIC							

U 063D, 452A,15	:	5226	BIC LS[00000000] TO WR[0]	:	CLEAR UPPER BYTE OF RESULT (UPPER BYTE
	:	5227		:	IS NOT PART OF TB AND IS UNPREDICT-
	:	5228		:	ABLE)
U 063E, 456E,15	:	5229	BIC LS[BIT23] TO WR[0]	:	DITTO FOR BIT 23
U 063F, 4540,15	:	5230	BIC LS[BIT0] TO WR[0]	:	DITTO FOR BIT 0. WR 0 NOW CONTAINS
	:	5231		:	RESULT TO PRINT
U 0640, BE0C,15	:	5232	MOV WR[0] TO LS[T6.SUB]	:	RESULT NOW IN LS 6
U 0641, 8868,74	:	5233	JMP [WAIT.SUB]	:	JUMP TO SET ATTN AND WAIT

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

DEPOSIT UNIBUS MAP 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
DEPOSIT UNIBUS MAP ROUTINE

H 10

Fiche 1 Frame H10

Sequence 124
Rev 01.00

Page 118

:5234 .PAGE " DEPOSIT UNIBUS MAP ROUTINE"

:5235 :+++

:5236

:5237

:5238

:5239

:5240

:5241

:5242

:5243

:5244

:5245

:5246

:5247

:5248

:5249

:5250

:5251

:5252

:5253

:5254

:5255

:5256

:5257

:5258

:5259

:5260

:5261

:5262

:5263

:5264

:5265

:5266

:5267

:5268

:5269

:5270

:5271

:5272

:5273

:5274

:5275

:5276

:5277

:5278

:5279

:5280

:5281

:5282

:5283

:5284

:5285

:5286

:5287

:5288

FUNCTIONAL DESCRIPTION:

This routine allows a write to the UNIBUS map portion of the TB on the memory controller. The map area contains 512 decimal locations addressed from 200-3FF. The remainder of the TB RAM is either unused or contains the Translation Buffer information. The DEPOSIT TB command is used to write the TB portion of the TB RAMs. The address specified with the DEPOSIT command will be the actual RAM address accessed. This routine will do any shifting necessary on the address specified to position it correctly. The address specified has been loaded by the console in LS location 5 prior to executing this routine. The address must be in HEX format.

The data specified has been placed in LS 6 by the console prior to executing this routine. Bits 07-01 will be the Valid, Prot, Modify, and Byte Offset bits (there are four Prot bits). Bits 22-08 will contain the PA bits from PA 23 through PA 09 respectively. Bits 31 through 23 and bit 0 are unused. This routine will do any shifting necessary to write the bits in TB as described. The data specified must be in HEX format.

CALLING SEQUENCE:

The console loads the address of a pointer to this routine (a JMP instruction at WCS location 58) into the UPC and starts the CPU.

INPUT PARAMETERS:

LS 5 - UNIBUS Map address in TB (must be in range 200-3FF HEX).
LS 6 - Contains the data to write into the UNIBUS Map in bits 1-22.

IMPLICIT INPUTS:

None

OUTPUT PARAMETERS:

The UNIBUS map portion of the TB RAMs gets written with the data from LS 6 at the address specified in LS 5.

IMPLICIT OUTPUTS:

None

SIDE EFFECTS:

WR 0 will be modified
WR 1 will be modified

DEPOSIT.UBS:

JSR [SHIFT.UB.ADR]

;GO TO SUBROUTINE TO ARRANGE UNIBUS

U 0642, 0864,DC

```

:5289
U 0643, 360C,15 :5290      MOV LS[T6.SUB] TO WR[0]      : MAP ADDRESS CORRECTLY
U 0644, 4526,15 :5291      BIC LS[#FF] TO WR[0]      : GET DATA FROM LS 6
:5292      : CLEAR LOW BYTE OF DATA SPECIFIED.
:5293      : LOW BYTE WILL BE PUT IN BITS 24-31
:5294      : LATER
U 0645, 8862,AC :5295      JSR [SHIFT.LOOP]      : GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
:5296      : PLACES RIGHT
U 0646, E40C,15 :5297      SWAP LS[T6.SUB] WITH WR[0]      : SWAP ORIGINAL DATA SPECIFIED WITH MOD-
:5298      : IFIED DATA. NOW LS 6 CONTAINS BITS
:5299      : 8-22 IN BITS 0-14 AS REQUIRED. WR 0
:5300      : CONTAINS ORIGINAL DATA SPECIFIED
U 0647, 452C,15 :5301      BIC LS[#FFFFFF00] TO WR[0]      : CLEAR ALL BUT LOW BYTE OF ORIGINAL
:5302      : DATA SPECIFIED
U 0648, 8862,AC :5303      JSR [SHIFT.LOOP]      : GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
:5304      : PLACES RIGHT. ON RETURN, WR 0 WILL
:5305      : HAVE BITS 0-7 IN BITS 24-31, OTHER
:5306      : BITS ARE CLEAR
U 0649, EDOC,15 :5307      BIS WR[0] TO LS[T6.SUB]      : NOW LS 6 CONTAINS DATA TO WRITE IN
:5308      : TB IN CORRECT FORMAT I.E. PA BITS
:5309      : IN BITS 00-14 AND BYTE OFFSET, MODIFY
:5310      : PROT A THROUGH PROT D, AND VALID IN
:5311      : BITS 25-31.
U 064A, 1B0B,F5 :5312      MEM.REQ[WRITE.UBS.MAP] ADRS[T5.SUB] DT[LONG] :ISSUE MEMORY REQUEST TO
:5313      : WRITE THE UNIBUS MAP
U 064B, B20C,15 :5314      WRITE.MEM LS[T6.SUB]      :WRITE DATA FROM LS 6 IN UNIBUS MAP
U 064C, 8868,74 :5315      JMP [WAIT.SUB]      :JUMP TO SET ATTN AND WAIT
:5316      :
:5317      : SUBROUTINE TO POSITION ADDRESS SPECIFIED CORRECTLY TO ADDRESS THE
:5318      : UNIBUS MAP.
:5319      :
:5320      : SHIFT.UB.ADR:
U 064D, 360A,15 :5321      MOV LS[T5.SUB] TO WR[0]      :GET ADDRESS SPECIFIED IN WR 0
U 064E, A2D0,15 :5322      SHL2 WR[0]      :SHIFT ADDRESS 2 PLACES LEFT
U 064F, A2D0,15 :5323      SHL2 WR[0]      :SHIFT ADDRESS 2 PLACES LEFT
U 0650, A2D0,15 :5324      SHL2 WR[0]      :SHIFT ADDRESS 2 PLACES LEFT
U 0651, A2D0,15 :5325      SHL2 WR[0]      :SHIFT ADDRESS 2 PLACES LEFT
U 0652, 23D0,15 :5326      ASHL WR[0]      :SHIFT LEFT ONE MORE PLACE
:5327      : NOW ADDRESS IS IN BITS 9-17 OF WR 0
:5328      : NOW LS 5 CONTAINS CORRECT ADDRESS.
U 0653, 3E0A,14 :5329      MOV WR[0] TO LS[T5.SUB],
:5330      : RETURN TO MAIN
:5331      : RETURN
  
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) EXAMINE UNIBUS MAP 3-MAR-82 09:34:33
EXAMINE UNIBUS MAP ROUTINE

J 10
3-MAR-82
Fiche 1 Frame J10
ENKCA Micro-diagnostic for DAP module

Sequence 126
Rev 01.00
Page 120

```
:5332 .PAGE " EXAMINE UNIBUS MAP ROUTINE"
:5333 .+++
:5334 This routine will read the UNIBUS map portion of the memory controller.
:5335 The UNIBUS MAP contents will be returned with the byte offset, modify,
:5336 prot A through prot D, and valid in bits 01-07 respectively. PA bits
:5337 09-23 are returned in bits 08-22 respectively. Bits 23-31 and bit 0
:5338 are not used and so will be cleared before printing the result. The
:5339 result will be put in LS 6 for the console to print.
:5340 The routine will issue a memory request with memory function=READ.UBS
:5341 .MAP and data type=LONGWORD. The console will load LS 5 with the ad-
:5342 dress specified before executing this routine. The routine will shift
:5343 the address around as required to address the UNIBUS map location spec-
:5344 ified. The address specified must be in HEX format.
:5345 The UNIBUS map consists of 512 decimal locations (addresses from 200-
:5346 3FF) The remaining locations in the TB RAM are either unused or used
:5347 by the translation buffer. TB contents are read via the EXAMINE TB
:5348 routine.
:5349
:5350 CALLING SEQUENCE:
:5351
:5352 The console loads the address of a pointer to this routine (a JMP in-
:5353 struction at WCS location 59) into the UPC and starts the CPU.
:5354
:5355 INPUT PARAMETERS:
:5356
:5357 LS 5 - TB address in hex format
:5358
:5359 IMPLICIT INPUTS:
:5360
:5361 None
:5362
:5363 OUTPUT PARAMETERS:
:5364
:5365 LS 6 - UNIBUS map data from address specified
:5366
:5367 IMPLICIT OUTPUTS:
:5368
:5369 None
:5370
:5371 SIDE EFFECTS:
:5372
:5373 LS 5 is modified
:5374 WR 0 will be modified
:5375
:5376 ---
:5377
:5378 EXAMINE.UBS:
:5379 JSR [SHIFT.UB.ADR] ;SHIFT ADDRESS SPECIFIED FROM LS 5 INTO
:5380 ; CORRECT POSITION AND REPLACE IN LS 5
:5381 MEM.REQ[READ.UBS.MAP] ADRS[TS.SUB] DT[LONG] ;ISSUE MEMORY REQUEST TO
:5382 ; READ THE UBS MAP AT ADDRESS CONTAINED
:5383 ; IN LS 5
:5384 MOV MEM.DATA TO WR[0] ;PLACE RESULT IN WR 0
:5385
:5386 BIC LS[0FF000000] TO WR[0] ;CLEAR UPPER BYTE OF RESULT (UPPER BYTE
```

U 0654, 0864,DC

U 0655, 1COB,75

U 0656, 3022,15

U 0657, 452A,15

U 0658, 456E,15	:5387		: IS NOT PART OF TB AND IS UNPREDICT-
U 0659, 4540,15	:5388		: ABLE)
	:5389	BIC LS[BIT23] TO WR[0]	:DITTO FOR BIT 23
	:5390	BIC LS[BIT0] TO WR[0]	:DITTO FOR BIT 0. WR 0 NOW CONTAINS
	:5391		: RESULT TO PRINT
U 065A, BE0C,15	:5392	MOV WR[0] TO LS[T6.SUB]	:RESULT NOW IN LS 6
U 065B, 8868,74	:5393	JMP [WAIT.SUB]	:JUMP TO SET ATTN AND WAIT
	:5394		: CONTROL

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) 3-MAR-82 09:34:33
DEPOSIT MAIN MEMORY ROUTINE

DEPOSIT MAIN MEMORY 3-MAR-82

Fiche 1 Frame L10

Sequence 128
Rev 01.00 Page 122

```
:5395 .PAGE  " DEPOSIT MAIN MEMORY ROUTINE"
:5396 .+++
:5397
:5398 .FUNCTIONAL DESCRIPTION:
:5399
:5400 .This routine write a longword of data into main memory at the Physical
:5401 .address specified. This address need not be on a longword boundary.
:5402 .The console will use this routine for data transfers to main memory as
:5403 .well as for the DEPOSIT MM command. The console will have loaded the
:5404 .address into LS 5 and the data into LS 6 before executing this routine.
:5405 .The routine issues a memory request with the memory function=WRITE.P
:5406 .and the data type=longword. It then issues a WRITE.MEM LS[6] instruc-
:5407 .tion to write the data into memory. A check on ERROR SUM is made to
:5408 .assure that the write occured without an error. If ERROR SUM is as-
:5409 .serted, the CPU will issue a CPU ACK before asserting CPU ATTN to in-
:5410 .form the console that an error occurred.
:5411
:5412 .CALLING SEQUENCE:
:5413
:5414 .The console loads the address of a pointer to this routine (a JMP in-
:5415 .struction at WCS location 54) into the UPC and starts the CPU.
:5416
:5417 .INPUT PARAMETERS:
:5418
:5419 .LS 5 - Main memory physical address
:5420 .LS 6 - Main memory longword data
:5421
:5422 .IMPLICIT INPUTS:
:5423
:5424 .None
:5425
:5426 .OUTPUT PARAMATERS:
:5427
:5428 .Main Memory address specified is written with data from LS 6.
:5429
:5430 .IMPLICIT OUTPUTS:
:5431
:5432 .A memory error asserts CPU ACK to inform the console of a write error.
:5433
:5434 .SIDE EFFECTS:
:5435
:5436 .None
:5437
:5438 .---
:5439
:5440 .DEPOSIT.MM:
:5441 .MEM.REQ[WRITE.P] ADRS[T5.SUB] DT[LONG] ;SET UP FOR WRITE TO MAIN
:5442 .MEMORY USING THE PHYSICAL ADDRESS
:5443 .STORED IN LS 5
:5444 .WRITE.MEM LS[T6.SUB], ;WRITE THE DATA FROM LS 6 INTO MAIN
:5445 .MEMORY
:5446 .SKIP.IF[MEM.REF.OK] ;SKIP NEXT INSTRUCTION IF NO MEMORY
:5447 .ERROR (NO ERR SUM)
:5448 .MISC [SET.CP.ACK] ;SET CPU ACK IF ERROR SUM ASSERTED
:5449
```

U 065C, 990A,75

U 065D, 320C,1D

U 065E, 10D0,15

U 065F, 3644,15 :5450	MOV LS[4] TO WR[0]	:PUT A 4 IN WR 0
U 0660, 6COA,15 :5451	ADD WR[0] TO LS[15.SUB]	:INCREMENT LS 5 FOR NEXT LONGWORD DEP
U 0661, 8868,74 :5452	JMP [WAIT.SUB]	:JUMP TO SET ATTN AND WAIT

```

:5453 PAGE " EXAMINE MAIN MEMORY ROUTINE"
:5454 ***
:5455
:5456 FUNCTIONAL DESCRIPTION:
:5457
:5458 This routine is used to read a longword from main memory at the Phy-
:5459 sical address specified. The address need not be on a longword bound-
:5460 ary. This routine is used by the console on an EXAMINE.MM command.
:5461 The console will have loaded LS 5 with the physical address specified
:5462 before executing this routine.
:5463 The routine will first write CSR 1 to set the INH REP CRD (inhibit
:5464 report correctable read data) bit to prevent an error sum from occurring
:5465 on a single bit error which has been corrected. Then the routine is-
:5466 sues a memory request with memory function=READ.P and DT=longword. It
:5467 then reads the memory location and puts the result in LS 6. It also
:5468 checks for an ERROR SUM and issues a CPU ACK if an error occurred (sin-
:5469 gle bit errors that have been corrected will not cause an ERR SUM). If
:5470 an error occurred, CPU ACK will be set before issuing CPU ATTN to in-
:5471 form the console that an error occurred.
:5472
:5473 CALLING SEQUENCE:
:5474
:5475 The console loads the address of a pointer to this routine (a JMP in-
:5476 struction at WCS location 55) into the UPC and starts the CPU.
:5477
:5478 INPUT PARAMETERS:
:5479
:5480 LS 5 - Physical address in main memory to read.
:5481
:5482 IMPLICIT INPUTS:
:5483
:5484 LS[INH.CRD] - Data to write into CSR 1 to inhibit error sum on a CRD.
:5485 LS[CSR1] - Address for writing CSR 1.
:5486
:5487 OUTPUT PARAMATERS:
:5488
:5489 LS 6 - Longword data from physical memory address specified.
:5490
:5491 IMPLICIT OUTPUTS:
:5492
:5493 CPU ACK if an error sum occurs.
:5494
:5495 SIDE EFFECTS:
:5496
:5497 None
:5498
:5499 ---
:5500
:5501 EXAMINE.MM:
:5502 MEM.REQ[WRITE.CSR] ADRS[CSR1] DT[LONG] :ISSUE MEMORY REQUEST TO
:5503 : WRITE CSR 1
:5504 WRITE.MEM LS[INH.CRD] :SET INH REP CRD IN CSR 1 AND CLEAR
:5505 : ECC DISABLE
:5506
:5507 MEM.REQ[READ.P] ADRS[T5.SUB] DT[LONG] :ISSUE MEMORY REQUEST TO

```

U 0662, 1D45,F5

U 0663, B278,15

U 0664, 180B,F5

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) EXAMINE MAIN MEMORY 3-MAR-82 09:34:33
EXAMINE MAIN MEMORY ROUTINE

B 11

Fiche 1 Frame B11

Sequence 131

Rev 01.00

Page 125

:5508
:5509
:5510
U 0665, 380C,1D :5511
:5512
U 0666, 10D0,15 :5513
U 0667, 8868,74 :5514

MOV MEM.DATA TO LS[T6.SUB],
SKIP.IF[MEM.REF.OK]

MISC [SET.CP.ACK]
JMP [WAIT.SUB]

: READ LONGWORD DATA FROM MAIN MEMORY
: PHYSICAL ADDRESS IN LS 5
: READ DATA AND PUT IN LS 6
: SKIP NEXT INSTRUCTION IF NO MEMORY
: ERROR (NO ERROR SUM)
: SET CPU ACK IF AN ERROR OCCURRED
: JUMP TO SET ATTN AND WAIT

5515 .page " EXAMINE IDC ROUTINES"
5516 :
5517 :
5518 :
5519 :
5520 :
5521 :
5522 :
5523 :
5524 :
5525 :
5526 :
5527 :
5528 :
5529 :
5530 :
5531 :
5532 :
5533 :
5534 :
5535 :
5536 :
5537 :
5538 :
5539 :
5540 :
5541 :
5542 :
5543 :
5544 :
5545 :
5546 :
5547 :
5548 :
5549 :
5550 :
5551 :
5552 :
5553 :
5554 :
5555 :
5556 :
5557 :
5558 :
5559 :
5560 :
5561 :
5562 :
5563 :
5564 :
5565 :
5566 :
5567 :
5568 :
5569 :

FUNCTIONAL DESCRIPTION:

The following routines are used to read data from the optional IDC module's registers. The result is put in LS 6 for the console to print out.

CALLING SEQUENCE:

The console loads the address of a pointer to this routine (a JMP instruction at WCS location 5A through 5E) into the UPC and starts the CPU.

INPUT PARAMETERS:

None

IMPLICIT INPUTS:

None

OUTPUT PARAMETERS:

LS 6 - Data from IDC register specified.

IMPLICIT OUTPUTS:

None

SIDE EFFECTS:

None

EXAMINE.ICSR:

```
MISC [SET.PORT.I.O]      :SELECT PORT TO BUS
PORT.READ PORT.ADRS[CSR]  :SEND THE READ COMMAND
JMP [READ.IDC]            :READ THE DATA
```

EXAMINE.IDAR:

```
MISC [SET.PORT.I.O]      :SELECT PORT TO BUS
PORT.READ PORT.ADRS[IDAR] :SEND THE READ COMMAND
JMP [READ.IDC]            :READ THE DATA
```

EXAMINE.DBUF:

```
MISC [SET.PORT.I.O]      :SELECT PORT TO BUS
PORT.READ PORT.ADRS[DATA.WORD] :SEND THE READ COMMAND
JMP [READ.IDC]            :READ THE DATA
```

EXAMINE.PATT:

```
MISC [SET.PORT.I.O]      :SELECT PORT TO BUS
PORT.READ PORT.ADRS[PATTERN] :SEND THE READ COMMAND
```

U 0668, 9090,15
U 0669, 9780,15
U 066A, 0867,64

U 066B, 9090,15
U 066C, 1784,15
U 066D, 0867,64

U 066E, 9090,15
U 066F, 978C,15
U 0670, 0867,64

U 0671, 9090,15
U 0672, 1790,15

U 0673, 0867,64	:5570	JMP [READ.IDC]	:READ THE DATA
	:5571		
	:5572	EXAMINE.POSIT:	
U 0674, 9090,15	:5573	MISC [SET.PORT.I.O]	:SELECT PORT TO BUS
U 0675, 9794,15	:5574	PORT.READ PORT.ADRS[POSITION]	:SEND THE READ COMMAND
	:5575		
	:5576		
	:5577	READ.IDC:	
U 0676, 3A0C,15	:5578	MOV PORT TO LS[T6.SUB]	:READ DATA AND PUT IN LS 6
U 0677, 1080,15	:5579	MISC [SET.ACC.I.O]	:RETURN SELECT TO ACCELERATOR
U 0678, 8868,74	:5580	JMP [WAIT.SUB]	:JUMP TO SET ATTN AND WAIT

:5581 .page " DEPOSIT IDC ROUTINES"
 :5582 +++
 :5583

:5584 FUNCTIONAL DESCRIPTION:
 :5585

:5586 The following routines are used to write data to the optional IDC
 :5587 module's registers. The data is taken from LS 6 which is previously
 :5588 loaded by the monitor when the DEPOSIT command is executed.
 :5589

:5590 CALLING SEQUENCE:
 :5591

:5592 The console loads the address of a pointer to this routine (a JMP in-
 :5593 struction at WCS location 5F through 61) into the UPC and starts the
 :5594 CPU.
 :5595

:5596 INPUT PARAMETERS:
 :5597

:5598 LS 6 - Data pattern to write.
 :5599

:5600 IMPLICIT INPUTS:
 :5601

:5602 None
 :5603

:5604 OUTPUT PARAMETERS:
 :5605

:5606 None
 :5607

:5608 IMPLICIT OUTPUTS:
 :5609

:5610 None
 :5611

:5612 SIDE EFFECTS:
 :5613

:5614 None
 :5615
 :5616 ---
 :5617

:5618 DEPOSIT.ICSR:
 :5619

:5619 MOV LS[T6.SUB] TO WR[0] :GET DATA PATTERN TO WRITE
 :5620 PORT.WRITE PORT.ADRS[CSR] WITH WR[0] :WRITE TO IDC
 :5621 JMP [WAIT.SUB] :JUMP TO SET ATTN AND WAIT
 :5622

:5623 DEPOSIT.IDAR:
 :5624

:5625 MOV LS[T6.SUB] TO WR[0] :GET DATA PATTERN TO WRITE
 :5626 PORT.WRITE PORT.ADRS[DAR] WITH WR[0] :WRITE TO IDC
 :5627 JMP [WAIT.SUB] :JUMP TO SET ATTN AND WAIT
 :5628

:5629 DEPOSIT.DBUF:
 :5630

:5631 MOV LS[T6.SUB] TO WR[0] :GET DATA PATTERN TO WRITE
 :5632 PORT.WRITE PORT.ADRS[DATA.WORD] WITH WR[0] :WRITE TO IDC
 :5633 JMP [WAIT.SUB] :JUMP TO SET ATTN AND WAIT
 :5634
 :5635

U 0679, 360C,15
 U 067A, 17A0,15
 U 067B, 8868,74

U 067C, 360C,15
 U 067D, 97A4,15
 U 067E, 8868,74

U 067F, 360C,15
 U 0680, 17AC,15
 U 0681, 8868,74

U 0682, 17C0,15	:5636	CLEAR.FIFO.ADDR:	
U 0683, 8868,74	:5637	PORT.CONTROL [CLEAR.FIFO.CNTR]	;CLEAR ADDRESS IN FIFO A OR FIFO B
	:5638	JMP [WAIT.SUB]	;JUMP TO SET ATTN AND WAIT
	:5639		
	:5640		
U 0684, 17D8,15	:5641	SELECT.FIFO.A:	
U 0685, 8868,74	:5642	PORT.CONTROL [SEL.FIFO.A]	;SELECT FIFO A
	:5643	JMP [WAIT.SUB]	;JUMP TO SET ATTN AND WAIT
	:5644		
	:5645		
U 0686, 97DC,15	:5646	SELECT.FIFO.B:	
	:5647	PORT.CONTROL [SEL.FIFO.B]	;SELECT FIFO B
	:5648		
	:5649		
U 0687, 10E0,15	:5650	WAIT.SUB:	
	:5651	MISC [SET.CP.ATTN]	;ISSUE CPU ATTN TO CONSOLE
U 0688, 8868,84	:5652	WAIT.DE.EX:	
		JMP [WAIT.DE.EX]	;LOOP SELF UNTIL CONSOLE TAKES CONTROL

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03)
SAVE WR ROUTINE

SAVE WR ROUTINE

3-MAR-82 09:34:33

G 11

3-MAR-82

Fiche 1 Frame G11

ENKCA Micro-diagnostic for DAP module

Sequence 136
Rev 01.00

Page 130

:5653 .PAGE " SAVE WR ROUTINE"

:5654 :+++

:5655

:5656

:5657

:5658

:5659

:5660

:5661

:5662

:5663

:5664

:5665

:5666

:5667

:5668

:5669

:5670

:5671

:5672

:5673

:5674

:5675

:5676

:5677

:5678

:5679

:5680

:5681

:5682

:5683

:5684

:5685

:5686

:5687

:5688

:5689

:5690

:5691

:5692

:5693

:5694

:5695

:5696

:5697

:5698

:5699

:5700

:5701

:5702

:5703

FUNCTIONAL DESCRIPTION:

This routine saves WRs 0-3 in LS locations 1-4. When the console takes control of the CPU, it calls this routine so that the working registers can be used by the console. The WRs are restored by another routine called by the console before the CPU resumes execution (on a Continue command, for example).

This routine simply moves data from WR 0-3 to LS locations 1-4 and then issues a CPU ATTN to the console. It then loops on a JMP self until the console takes over.

CALLING SEQUENCE:

The console loads the address of a pointer to this routine (a JMP instruction at WCS location 46) into the UPC and starts the CPU.

INPUT PARAMETERS:

None

IMPLICIT INPUTS:

None

OUTPUT PARAMETERS:

LS 1 - Contents of WR 0
LS 2 - Contents of WR 1
LS 3 - Contents of WR 2
LS 4 - Contents of WR 3

IMPLICIT OUTPUTS:

None

SIDE EFFECTS:

None

SAVE.WR:

MOV WR[0] TO LS[SAV.WR0]
MOV WR[1] TO LS[SAV.WR1]
MOV WR[2] TO LS[SAV.WR2]
MOV WR[3] TO LS[SAV.WR3]
JMP [WAIT.SUB]

:SAVE WR 0 IN LS LOCATION 1
:SAVE WR 1 IN LS LOCATION 2
:SAVE WR 2 IN LS LOCATION 3
:SAVE WR 3 IN LS LOCATION 4
:JUMP TO SET ATTN AND WAIT

U 0689, 3E02,15
U 068A, BE04,95
U 068B, 3E07,15
U 068C, 3E09,95
U 068D, 8868,74

:5704 .PAGE " RESTORE WR ROUTINE"
 :5705 +++

:5706 FUNCTIONAL DESCRIPTION:

:5707 This routine restores the 2901 working registers save in LS 1-4. The
 :5708 console calls this routine before the CPU is restarted after it has
 :5709 been halted by the console.
 :5710 The routine simply moves the data from LS locations 1-4 to WRs 0-3
 :5711 and then issues a CPU ATTN to the console. A JMP self is executed next
 :5712 waiting for the console to take control.
 :5713
 :5714
 :5715

:5716 CALLING SEQUENCE:

:5717 The console loads the address of a pointer to this routine (a JMP in-
 :5718 struction at WCS location 47) into the UPC and starts the CPU.
 :5719
 :5720

:5721 INPUT PARAMETERS:

:5722 LS 1 contains the data to be restored in WR 0.
 :5723 LS 2 contains the data to be restored in WR 1.
 :5724 LS 3 contains the data to be restored in WR 2.
 :5725 LS 4 contains the data to be restored in WR 3.
 :5726
 :5727

:5728 IMPLICIT INPUTS:

:5729 None

:5730 OUTPUT PARAMATERS:

:5731 WR 0 gets data from LS 1.
 :5732 WR 1 gets data from LS 2.
 :5733 WR 2 gets data from LS 3.
 :5734 WR 3 gets data from LS 4.
 :5735
 :5736
 :5737

:5738 IMPLICIT OUTPUTS:

:5739 None

:5740 SIDE EFFECTS:

:5741 None

:5742 ---

:5743 RESTORE.WR:

U 068E, B602.15	:5750	MOV LS[SAV.WR0] TO WR[0]	:RESTORE WR 0
U 068F, 3604.95	:5751	MOV LS[SAV.WR1] TO WR[1]	:RESTORE WR 1
U 0690, B607.15	:5752	MOV LS[SAV.WR2] TO WR[2]	:RESTORE WR 2
U 0691, B609.95	:5753	MOV LS[SAV.WR3] TO WR[3]	:RESTORE WR 3
U 0692, 8868.74	:5754	JMP [WAIT.SUB]	:JUMP TO SET ATTN AND WAIT


```

:5755 .PAGE 'WCS SUBROUTINES FOR CPU USE'
:5756 .TOC 'ERROR ROUTINE'
:5757 .+++
:5758
:5759 .FUNCTIONAL DESCRIPTION:
:5760
:5761 This routine is used to detect and report diagnostic errors occurring
:5762 during WCS based micro-diagnostics. The WCS micro-code sets up the
:5763 parameters listed below and calls this routine. The routine compares
:5764 WR 1 (expected data) and WR 0 (received data) according to the error
:5765 mask. Bits set in the mask indicate bits that are not checked for
:5766 errors.
:5767 If WR 0 and WR 1 are equal (no error) the routine executes a return+1
:5768 to the calling point. The only exception to this is if loop on error
:5769 is set. In that case, the error number is checked to see if that error
:5770 occurred previously. If so, then the error loop is forced by doing a
:5771 return. This will lock an error loop in on intermittent errors.
:5772 If WR 0 and WR 1 are not equal (an error) then bit 8 is set in the
:5773 CONTROL AND STATUS word, expected and received data is set up in LS for
:5774 printout, and flags are checked in the ERROR CONTROL word. This con-
:5775 trols whether printouts are disabled, loop on error is enabled, etc.
:5776 After printing the error, a return+1 is made to the calling point.
:5777 Loop on error causes a return rather than return+1 to cause an error
:5778 loop. Also CPU ACKNOWLEDGE is toggled at the end of the routine for
:5779 a scope sync point.
:5780
:5781 .CALLING SEQUENCE:
:5782
:5783 JSR [CHECK.RESULT]
:5784
:5785 .INPUT PARAMETERS:
:5786
:5787 WR[0] = Received data i.e. the actual result of the test
:5788 WR[1] = Expected data i.e. what the result should have been
:5789
:5790 .IMPLICIT INPUTS:
:5791
:5792 LS[ERROR.MASK] = Used to mask out bits that are not to be tested
:5793 LS[ERROR.NUMBER] = Current error number
:5794 LS[PREVIOUS.ERROR] = Error number of previous data
:5795 LS[CONTROL.STATUS] = Control and Status word. Contains information
:5796 written by the CPU to inform the monitor what to do.
:5797 LS[ERROR.CONTROL] = Information such as loop-on-error, bell-on-error,
:5798 no error report, and halt on error.
:5799
:5800 .OUTPUT PARAMATERS:
:5801
:5802 NONE
:5803
:5804 .IMPLICIT OUTPUTS:
:5805
:5806 LS[CONTROL.STATUS] = Control and Status with new status information
:5807 LS[PREVIOUS.ERROR] = Last error number (modified only if error and
:5808 loop on error is enabled)
:5809 LS[EXPECTED.DATA] = Expected result if an error was detected

```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03)
ERROR ROUTINE

ERROR ROUTINE

3-MAR-82 09:34:33

J 11

3-MAR-82

Fiche 1 Frame J11

Sequence 139

Rev 01.00

Page 133

```
:5810 : LS[RECEIVED.DATA] = Actual result if an error was detected
:5811 :
:5812 : SIDE EFFECTS:
:5813 :
:5814 : WR[0], WR[1], and the Q reg are modified and are not restored before
:5815 : returning.
:5816 : If loop-on-error is enabled and an error loop is to be executed, CPU
:5817 : ACKNOWLEDGE will be toggled for a scope sync.
:5818 :
:5819 : ---
:5820 :
:5821 : CHECK.RESULT:
U 0693, 458A,15 :5822 BIC LS[ERROR.MASK] TO WR[0] :MASK OFF NOT TESTED BITS (CLEAR THEM)
:5823 : FOR RECEIVED DATA
U 0694, C58A,95 :5824 BIC LS[ERROR.MASK] TO WR[1] :DITTO FOR EXPECTED DATA
:5825 :
:5826 : XOR WR[0] WITH WR[1] TO Q, :CMP RECEIVED DATA IN WR[0] WITH
:5827 : DT(LONG)&SET.ALU.CC : EXPECTED DATA IN WR[1] FOR ERROR
U 0696, 8869,F1 :5828 JMP.IF[NEQ] TO [ERROR] :JUMP IF ERROR
:5829 :
:5830 : MOV LS[ERROR.CONTROL] TO WR[0] :GET ERROR CONTROL WORD FROM LS
:5831 : BIT WR[0] WITH LS[LOE], :SEE IF LOOP ON ERROR IS ENABLED
:5832 : DT(LONG)&SET.ALU.CC : SET CONDITION CODES
U 0698, 5940,35 :5833 SKIP.IF[BIT.SET] :SKIP IF IT IS
U 0699, DB00,01 :5834 RETURN+1 :ELSE RETURN+1 (NO ERROR AND NO LOOP)
U 069A, DB00,16 :5835 :
:5836 : MOV LS[PREVIOUS.ERROR] TO WR[0] :IF NO ERROR BUT LOOP ON ERROR IS
:5837 : : ENABLED, SEE IF THERE WAS A
:5838 : : PREVIOUS ERROR
:5839 : XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,
U 069C, CD82,35 :5840 : DT(LONG)&SET.ALU.CC :IS THIS THE SAME SPOT THAT HAD AN
:5841 : : ERROR BEFORE? (INTERMITTANT ERROR)
U 069D, 086B,91 :5842 JMP.IF[NEQ] TO [RET+1] :JUMP IF NOT TO RETURN WITHOUT ERROR
:5843 : : LOOPING (RETURN+1)
U 069E, 086B,C4 :5844 JMP [RET] :ELSE CONTINUE TO LOOP (WE WILL LOOP
:5845 : : ON ERROR EVEN IF IT GOES AWAY)
:5846 :
:5847 : ERROR:
U 069F, 3E86,15 :5848 MOV WR[0] TO LS[RECEIVED.DATA] :PUT RESULT OF TEST IN LS FOR PRINTOUT
U 06A0, 3690,15 :5849 MOV LS[ERROR.CONTROL] TO WR[0] :GET ERROR CONTROL BITS TO CHECK FOR
:5850 : : LOOP COMMAND
:5851 : BIT WR[0] WITH LS[LOOP.COMMAND], :SEE IF BIT 6 SET
:5852 : DT(LONG)&SET.ALU.CC : SET CONDITION CODES
U 06A1, 594C,35 :5853 JMP.IF[BIT.SET] TO [RET] :GO LOOP IF SET (FAST LOOP)
:5854 :
:5855 : MOV WR[1] TO LS[EXPECTED.DATA] :PUT EXPECTED DATA IN LS FOR PRINTOUT
:5856 :
:5857 : MOV LS[CONTROL.STATUS] TO WR[1] :GET CONTROL AND STATUS WORD FROM LS
U 06A4, 3680,95 :5858 BIC LS[#FE7FFFFFFF] TO WR[1] :CLEAR ALL BUT BITS 23&24 IN CONTROL
U 06A5, C532,95 :5859 : AND STATUS WORD
:5860 : BIS LS[ERROR] TO WR[1] :SET BIT 8 IN CONTROL AND STATUS WORD
:5861 : : (INDICATES AN ERROR WAS DETECTED)
U 06A7, BE80,95 :5862 MOV WR[1] TO LS[CONTROL.STATUS] :WRITE UPDATED CONTROL AND STATUS WORD
:5863 : : BACK TO LS[40]
:5864 :
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03)
ERROR ROUTINE

ERROR ROUTINE

3-MAR-82 09:34:33

K 11
3-MAR-82

Fiche 1 Frame K11

ENKCA Micro-diagnostic for DAP module

Sequence 140
Rev 01.00

Page 134

```
:5865      BIT WR[0] WITH LS[BELL],      ;SEE IF BELL ON ERROR IS SET (WR 0 STILL
:5866      DT(LONG)&SET.ALU.CC          ;CONTAINS ERROR CONTROL WORD)
U 06A8, D944,35 :5867      JMP.IF[BITS.CLR] TO [CHECK.NER] ;SET CONDITION CODES
U 06A9, 886A,D9 :5868      MISC [SET.CP.ATTN]      ;IF BELL ON ERROR IS NOT SET, JUMP TO
:5869      ;SEE IF ERROR REPORTING IS INHIBITED
U 06AA, 10E0,15 :5870      WAIT.1: JMP [WAIT.1]      ;ELSE SET CPU ATTN (RING MY BELL)
:5871      JMP [LOOP]
U 06AB, 086A,B4 :5872      ;WAIT FOR CONSOLE TO STOP ME
U 06AC, 086B,64 :5873      ;GO TO CHECK LOOP-ON-ERROR AFTER
:5874      ;CONSOLE HAS FINISHED
:5875
:5876      CHECK.NER:
:5877      BIT WR[0] WITH LS[NER],      ;IF NO BELL SEE IF ERROR REPORT IS
:5878      DT(LONG)&SET.ALU.CC          ;INHIBITED
U 06AD, D942,35 :5879      JMP.IF[BIT.SET] TO [CHECK.HALT] ;SET CONDITION CODES
U 06AE, 886B,21 :5880      ;JUMP IF ERROR REPORT IS INHIBITED TO
:5881      ;CHECK FOR HALT ON ERROR
:5882
U 06AF, 10E0,15 :5883      MISC [SET.CP.ATTN]      ;SET CPU ATTN (REPORT ERROR)
U 06B0, 086B,04 :5884      WAIT.2: JMP [WAIT.2]      ;WAIT FOR CONSOLE TO STOP ME
U 06B1, 086B,64 :5885      JMP [LOOP]      ;GO TO CHECK LOOP-ON-ERROR AFTER
:5886      ;CONSOLE HAS FINISHED
:5887
:5888      CHECK.HALT:
:5889      BIT WR[0] WITH LS[HOE],      ;SEE IF HALT ON ERROR IS ENABLED
:5890      DT(LONG)&SET.ALU.CC          ;SET CONDITION CODES
U 06B2, 5946,35 :5891      JMP.IF[BITS.CLR] TO [LOOP]      ;JUMP IF NOT TO CHECK LOOP-ON-ERROR
:5892
U 06B4, 10E0,15 :5893      MISC [SET.CP.ATTN]      ;ELSE SET CPU ATTN (HALT ON ERROR)
U 06B5, 086B,54 :5894      WAIT.3: JMP [WAIT.3]      ;WAIT FOR CONSOLE TO STOP ME
:5895
U 06B6, 3690,15 :5896      LOOP: MOV LS[ERROR.CONTROL] TO WR[0] ;GET ERROR CONTROL BITS (NER, HOE ETC.)
:5897      BIT WR[0] WITH LS[LOE],      ;SEE IF LOOP-ON-ERROR
:5898      DT(LONG)&SET.ALU.CC          ;SET CONDITION CODES
U 06B7, 5940,35 :5899      SKIP.IF[BIT.SET]      ;SKIP IF LOOP ON ERROR IS ENABLED
:5900
U 06B9, DB00,16 :5901      RET+1: RETURN+1      ;ELSE RETURN+1 FOR NO ERROR LOOP
:5902
U 06BA, 3682,15 :5903      MOV LS[ERROR.NUMBER] TO WR[0] ;GET ERROR NUMBER IF LOOPING
:5904
U 06BB, BEA0,15 :5905      MOV WR[0] TO LS[PREVIOUS.ERROR] ;AND SAVE IT IN PREVIOUS ERROR
:5906
U 06BC, 10D0,15 :5907      RET: MISC [SET.CP.ACK]      ;SET CPU ACKNOWLEDGE FOR SCOPE SYNC
:5908      MISC [CLR.CP.ATTN.AND.ACK], ;AND THEN CLEAR IT
:5909      RETURN      ;RETURN TO TEST AND LOOP ON ERROR
:5910
```



```

:5911 .PAGE
:5912
:5913
:5914
:5915 ERR.NO.TEST:
U 06BE, DB00,15 :5916 NOP ;NOP WILL CAUSE A SEQUENCE ERROR
U 06BF, B65E,15 :5917 MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT 15 IN WR[0] FOR CONTROL AND
:5918 ;STATUS WORD
U 06C0, 3E80,15 :5919 MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
:5920 ;BIT 15 INDICATES START OF TEST TO
:5921 ;CONSOLE
U 06C1, 8868,74 :5922 JMP [WAIT.SUB] ;JUMP TO SET ATTN AND WAIT
:5923 "START TRANSFER ROUTINE"
:5924
:5925 .TOC
:5926 +++
:5927 This routine loads the LS with information necessary for these tests
:5928 to run. It will then issue a CPU ATTN with the control and status word
:5929 clear to indicate that all transfers are complete.
:5930
:5931 RESULT: LS LOADED WITH CONSTANTS. MONITOR INFORMED TRANSFERS OVER.
:5932
:5933 .---
:5934 .list
:5935 TRANSFER.POINT:
U 06C2, 2F80,15 :5935 CLR WR[0] ;START WITH 0 IN WR[0]
U 06C3, BFFE,15 :5936 MOV WR[0] TO LS[PSL.HW] ;MAKE SURE COMPAT MODE IS CLEAR
:5937
U 06C4, 2040,15 :5938 INC WR[0] ;NOW HAVE 1 IN WR[0]
U 06C5, 23D0,15 :5939 ASHL WR[0] ;10(B)
U 06C6, 2040,15 :5940 INC WR[0] ;11(B)
U 06C7, 23D0,15 :5941 ASHL WR[0] ;110(B)
U 06C8, 2040,15 :5942 INC WR[0] ;111(B)
U 06C9, 23D0,15 :5943 ASHL WR[0] ;1110(B)
U 06CA, 23D0,15 :5944 ASHL WR[0] ;1 1100(B)
U 06CB, 23D0,15 :5945 ASHL WR[0] ;11 1000(B)
U 06CC, 23D0,15 :5946 ASHL WR[0] ;111 0000(B) NOW HAVE 70(H) IN WR 0.
:5947 ;THIS IS CORRECT START ADDRESS OF DATA
:5948 ;SECTION (INCLUDING LONGWORD COUNT,
:5949 ;DESTINATION, AND DESTINATION ADDRESS)
U 06CD, 3E0E,15 :5950 MOV WR[0] TO LS[TRANSFER.POINTER] ;LOAD LS 7 WITH POINTER TO TRANSFER
:5951 ;DATA SECTION.
:5952
U 06CE, 2F80,15 :5953 CLR WR[0] ;0 IN WR 0
U 06CF, 2040,15 :5954 INC WR[0] ;1 IN WR 0
U 06D0, 23D0,15 :5955 ASHL WR[0] ;10(B)
U 06D1, 23D0,15 :5956 ASHL WR[0] ;100(B)
U 06D2, 23D0,15 :5957 ASHL WR[0] ;1000(B)
U 06D3, 23D0,15 :5958 ASHL WR[0] ;1 0000(B)
U 06D4, 23D0,15 :5959 ASHL WR[0] ;10 0000(B)
U 06D5, 23D0,15 :5960 ASHL WR[0] ;100 0000(B)
U 06D6, 23D0,15 :5961 ASHL WR[0] ;1000 0000(B)
U 06D7, 23D0,15 :5962 ASHL WR[0] ;1 0000 0000(B)
U 06D8, 23D0,15 :5963 ASHL WR[0] ;10 0000 0000(B)
U 06D9, 23D0,15 :5964 ASHL WR[0] ;100 0000 0000(B)
U 06DA, 23D0,15 :5965 ASHL WR[0] ;1000 0000 0000(B)

```

ZZ-ENKCA-1.0	:	ENKCA.MIC							
:	:	ENKCA.MCR	MICRO2 1L(03)	START TRANSFER ROUT	3-MAR-82	09:34:33	M 11	Fiche 1	Frame M11
:	:	ENKCA.MIC	START TRANSFER ROUTINE				ENKCA Micro-diagnostic for DAP module	Sequence 142	Page 136
								Rev 01.00	

U 06DB, 23D0,15	:	5966	ASHL WR[0]	:	1 0000 0000 0000(B)
U 06DC, 23D0,15	:	5967	ASHL WR[0]	:	10 0000 0000 0000(B)
U 06DD, 3E80,15	:	5968	MOV WR[0] TO LS[CONTROL.STATUS]	:	SET BIT 13 IN LS CONTROL AND STATUS
	:	5969		:	WORD (BIT 13 INDICATES CONSOLE MUST
	:	5970		:	PERFORM A DATA TRANSFER)
U 06DE, 10E0,15	:	5971	MISC [SET.CP.ATTN]	:	ISSUE CPU ATTN TO CONSOLE (DO FIRST LS
	:	5972		:	TRANSFER TO LOCATIONS 0 THROUGH 77(H))
	:	5973	WAIT.XFER1:		
U 06DF, 086D,F4	:	5974	JMP [WAIT.XFER1]	:	LOOP HERE WAITING FOR CONSOLE TO RE-
	:	5975		:	SPOND
	:	5976			
U 06E0, 36CA,15	:	5977	MOV LS[#162(H)] TO WR[0]	:	GET START ADDRESS OF SECOND HALF OF LS
	:	5978		:	TRANSFER
U 06E1, 3E0E,15	:	5979	MOV WR[0] TO LS[TRANSFER.POINTER]	:	LOAD START ADDRESS IN LS FOR CONSOLE
U 06E2, 365A,15	:	5980	MOV LS[XFER.DATA] TO WR[0]	:	BIT 13 INDICATES TO DO DATA TRANSFER
U 06E3, 3E80,15	:	5981	MOV WR[0] TO LS[CONTROL.STATUS]	:	SET UP CONTROL AND STATUS WORD
U 06E4, 10E0,15	:	5982	MISC [SET.CP.ATTN]	:	ISSUE CPU ATTN TO CONSOLE (DO SECOND LS
	:	5983		:	TRANSFER TO LOCATIONS 80 THROUGH F7(H))
	:	5984	WAIT.XFER2:		
U 06E5, 086E,54	:	5985	JMP [WAIT.XFER2]	:	LOOP HERE WAITING FOR CONSOLE TO RE-
	:	5986		:	SPOND
	:	5987			
U 06E6, B724,15	:	5988	MOV LS[HI.IPL] TO WR[0]		
U 06E7, BFFE,15	:	5989	MOV WR[0] TO LS[PSL.HW]	:	SET IPL TO 1F
	:	5990			
U 06E8, 2F80,15	:	5991	CLR WR[0]	:	CLEAR ALL BITS OF CONTROL AND STATUS
U 06E9, 3E80,15	:	5992	MOV WR[0] TO LS[CONTROL.STATUS]	:	WORD
	:	5993			
U 06EA, 10E0,15	:	5994	MISC [SET.CP.ATTN]	:	CALL CONSOLE
	:	5995	WAIT.XFER:		
U 06EB, 886E,B4	:	5996	JMP [WAIT.XFER]	:	WAIT FOR CONSOLE TO RESPOND
U 06EC, 5B00,14	:	5997	RETURN	:	USED WHEN TRANSFER IS CALLED FROM
	:	5998		:	MICRO-DIAGNOSTIC ONLY
	:	5999	.LIST		

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03)
START TRANSFER ROUTINE

START TRANSFER ROUT 3-MAR-82 09:34:33
N 11

ENKCA Micro-diagnostic for DAP module

Fiche 1 Frame N11

Sequence 143
Rev 01.00

Page 137

:6000
:6001
:6002

.PAGE
.REGION/700,3FFF


```

:6003 .PAGE "General Use Subroutines"
:6004 MOV.EX.RE:
U 0700, 3E86,15 :6005     MOV WR[0] TO LS[RECEIVED.DATA] ;SET UP RECEIVED
U 0701, 3E84,95 :6006     MOV WR[1] TO LS[EXPECTED.DATA] ; AND EXPECTED DATA
U 0702, 3682,15 :6007     MOV LS[ERROR.NUMBER] TO WR[0] ;GET ERROR NUMBER
U 0703, BEA0,15 :6008     MOV WR[0] TO LS[PREVIOUS.ERROR] ;PUT IT IN PREVIOUS ERROR
U 0704, 5B00,14 :6009     RETURN
:6010 INC.EN:
U 0705, 3682,15 :6011     MOV LS[ERROR.NUMBER] TO WR[0] ;GET ERROR NUMBER
U 0706, 2040,15 :6012     INC WR[0] ; AND INCREMENT
U 0707, BE82,15 :6013     MOV WR[0] TO LS[ERROR.NUMBER] ; IT.
U 0708, 5B00,14 :6014     RETURN
:6015 ISSUE.SS:
U 0709, 10D0,15 :6016     MISC [SET.CP.ACK] ;SCOPE SYNC POINT
U 070A, 90C0,15 :6017     MISC [CLR.CP.ATTN.AND.ACK] ;CLEAR SIGNALS FOR SYNCING
U 070B, 5B00,14 :6018     RETURN
:6019 INC.OS:
U 070C, 36F8,95 :6020     MOV LS[OS] TO WR[1] ;GET THE INDEX
U 070D, C52C,95 :6021     BIC LS[FFFFFFF0] TO WR[1] ;CLEAR HIGH BITS
U 070E, 2042,95 :6022     INC WR[1] ;INCREMENT
U 070F, BEF8,95 :6023     MOV WR[1] TO LS[OS] ;PUT IT BACK
U 0710, 5B00,14 :6024     RETURN
:6025 DEC.OS:
U 0711, 36F8,95 :6026     MOV LS[OS] TO WR[1] ;GET THE INDEX
U 0712, C52C,95 :6027     BIC LS[FFFFFFF0] TO WR[1] ;CLEAR HIGH BITS
U 0713, 4140,95 :6028     SUB LS[#1] FROM WR[1] ;DECREMENT
U 0714, BEF8,95 :6029     MOV WR[1] TO LS[OS] ;PUT IT BACK
U 0715, 5B00,14 :6030     RETURN
:6031 TEST.PREV:
U 0716, 3682,15 :6032     MOV LS[ERROR.NUMBER] TO WR[0]
:6033     XOR WR[0] WITH LS[PREVIOUS.ERROR] TO Q,
:6034     DT(LONG)&SET.ALU.CC
U 0717, CDA0,35 :6035     SKIP.IF[EQL]
U 0718, 5B00,09 :6036     RETURN+1
U 0719, DB00,16 :6037     RETURN
U 071A, 5B00,14 :6037
  
```

:6038
:6039
:6040
:6041
:6042
:6043
:6044
:6045
:6046
:6047
:6048
:6049
:6050
:6051
:6052
:6053
:6054
:6055
:6056
:6057
:6058
:6059
:6060
:6061
:6062
:6063
:6064
:6065
:6066
:6067
:6068
:6069
:6070
:6071
:6072
:6073
:6074
:6075
:6076
:6077
:6078
:6079
:6080
:6081
:6082
:6083
:6084
:6085
:6086
:6087
:6088
:6089
:6090
:6091
:6092

Page "TEST 1 - Unconditional Skip Test (M8390 CPU Module)"

++

Test Description:

This test checks the unconditional skip logic and associated increment micro-address logic. The test will check that every micro-address bit can be incremented correctly when a skip condition occurs. The data path is as follows: SCTL bits 0-4 set up the SEQ.CTL PAL to do an unconditional skip (assert pin 19 low). The PAL will output to the 3 input negated OR gate which controls INCR CIN H. This output goes to the adder which controls whether BUS uPC is incremented or not. The result is sent through the 2 to 1 MUX resulting in NAD 00 L through NAD 03 L. This output goes directly to the uPC PAL. NAD 04 through NAD 14 going to the uPC PALs come from the 2 to 1 MUX which has BUS NAD D04 through BUS NAD D10 and BUS uPC D11 through BUS uPC D14 as the selected input. BUS NAD D04 through BUS NAD D10 come from the INCR.HI PAL. LOW INCR COUT H from the adder controls the increments in this PAL.

Each micro-address bit is checked by executing the skip at specific micro-addresses which cause the increment to ripple through the address bits. For example, if the skip is executed at address 7, the increment logic must ripple through the first 3 bits to form an 8. Micro-address bits 0 through 10 are checked in this manner. After this increment, the micro-sequencer will increment the address again to cause the skip. Branching is not possible over the micro-address bit 10 boundary.

Assumptions:

It is assumed that the basic u-address sequencer has been checked previously or is known to be good. This is the first test of the skip logic and the first test to execute directly out of WCS at speed. One other assumption is that the MUX&INCR CTL PAL will not cause an unexpected skip.

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Execute a NOP with SCTL set to SKIP unconditionally (1E).
- 3) If no skip, report error. Else repeat step 2 at a new micro-address. The micro-addresses are picked so that INCR CIN H will cause the increment to ripple through all bits.
- 4) Test complete when micro-address bits 0 through 10 have been checked individually.

Errors:

- Error 1 - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from 0 to 1.
- Error 2 - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from 1 to 2.
- Error 3 - SCTL=skip failed to cause a skip. Address bits 0-10 should

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 1 - Unconditional Skip Test (M8390 CPU Module)
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 1 - Unconditional Skip Test (M8390 CPU Module)

Fiche 1 Frame D12
ENKCA Micro-diagnostic for DAP module

Sequence 146
Rev 01.00 Page 140

:6093
:6094
:6095
:6096
:6097
:6098
:6099
:6100
:6101
:6102
:6103
:6104
:6105
:6106
:6107
:6108
:6109
:6110
:6111
:6112
:6113
:6114
:6115
:6116
:6117
:6118
:6119
:6120
:6121
:6122
:6123
:6124
:6125
:6126
:6127
:6128
:6129
:6130
:6131
:6132
:6133
:6134
:6135
:6136
:6137
:6138
:6139
:6140
:6141
:6142
:6143
:6144
:6145
:6146
:6147

Error 4 - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from 3 to 4.
Error 5 - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from 7 to 8.
Error 6 - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from F to 10.
Error 7 - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from 1F to 20.
Error 8 - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from 3F to 40.
Error 9 - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from 7F to 80.
Error A - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from FF to 100.
Error B - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from 1FF to 200.
Error B - SCTL=skip failed to cause a skip. Address bits 0-10 should have incremented from 3FF to 400.

Debug:

Error 1 - This will probably be the most likely error to occur, since a lot of previously untested logic is involved. The first signal to examine is INCR CIN H at the adder chip. This signal should go high when the instruction with SCTL=skip is executed. If the signal fails to go high the problem is either in the SEQ.CTL PAL, it's inputs, or in the 3 input negated 'OR' gate. CSR bits 0-4 into the PAL should equal 1E(H) and JUMP INST L should be high. These inputs should produce a low on the output pin 19. The low should force a high out of the negated 'OR' gate. If INCR CIN H is ok, then suspect the adder chip.

Error 2-B - If error 1 did not occur, any of these errors indicates a problem with the adder (for errors 2-4) or the INCR.HI PAL or it's LOW INCR COUT H input (for errors 5-11).

U 071B, B65E,15
U 071C, 3E80,15
U 071D, 10E0,15
U 071E, 0871,E4
U 071F, 2F80,15
U 0720, 3E8A,15
U 0721, 2040,15
U 0722, BE82,15
U 0723, 2040,15
U 0724, 3E8C,15
U 0725, 2F80,15
U 0726, BEA0,15
U 0727, 363C,15
U 0728, 3E80,15

T.1:

WAIT.T1.0:

MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN] ;BIT15 INDICATES START OF TEST TO CONSOLE
;ISSUE CPU ATTN TO CONSOLE
JMP [WAIT.T1.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR WR[0] ;SET WRO TO 0
MOV WR[0] TO LS[ERROR.MASK] ;CLEAR ERROR MASK. MASK IS NOT USED IN
;THIS TEST SINCE WE ARE CHECKING SKIPS
INC WR[0] ;SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
INC WR[0] ;SET WRO TO 2
MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
CLR WR[0] ;SET WRO TO 0
MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
MOV LS[ERR.CON.NA] TO WR[0] ;LOAD WRO WITH A CONSTANT THAT SETS BITS 8,10, & 23
MOV WR[0] TO LS[CONTROL.STATUS] ;LOAD CONTROL AND STATUS WORD. BIT 8
; SET INDICATES AN ERROR, BIT 10 SET

U 0729, 097F.D4	:6148			: INDICATES CONSOLE CONTROLS ERROR
	:6149			: LOOPING, BIT 23 SET INDICATES THAT THE
	:6150			: CONSOLE SHOULD NOT PRINT EXPECTED OR
	:6151			: RECEIVED DATA ON ERROR REPORT. THIS
	:6152			: WORD IS LOADED NOW SO THAT IT NEEDN'T
	:6153			: BE LOADED AGAIN IF AN ERROR OCCURS.
	:6154	JMP [LOOP.T1.1]		: SET ADDRESS FOR FIRST SKIP
U 17FD, 10D0.15	:6155	17FD:		
U 17FE, 90C0.15	:6156	LOOP.T1.1:		: SCOPE SYNC POINT
	:6157	MISC [SET.CP.ACK]		: CLEAR CPU ACK FOR SYNCING
	:6158	MISC [CLR.CP.ATTN.AND.ACK]		: NOP
	:6159	NOP		: SHOULD SKIP NEXT INSTRUCTION
U 17FF, 5B00.1E	:6160	SKIP		: ERROR IF HERE
U 1800, 8874.04	:6161	JMP [ERROR.T1.1]		
	:6162			
U 1801, B682.95	:6163	T1.2:		: IF NO ERROR UPDATE ERROR NUMBER
U 1802, 2042.95	:6164	MOV LS[ERROR.NUMBER] TO WR[1]		:
U 1803, 3E82.95	:6165	INC WR[1]		:
U 1804, 887F.E4	:6166	MOV WR[1] TO LS[ERROR.NUMBER]		:
	:6167	JMP [LOOP.T1.2]		: SET ADDRESS FOR NEXT SKIP
	:6168			
U 07FE, 10D0.15	:6169	7FE:		
U 07FF, 90C0.15	:6170	LOOP.T1.2:		: SCOPE SYNC POINT
	:6171	MISC [SET.CP.ACK]		: CLEAR CPU ACK FOR SYNCING
	:6172	MISC [CLR.CP.ATTN.AND.ACK]		: NOP
U 0800, 5B00.1E	:6173	NOP		: SHOULD SKIP NEXT INSTRUCTION
U 0801, 0874.44	:6174	SKIP		: ERROR IF HERE
	:6175	JMP [ERROR.T1.2]		
U 0802, B682.95	:6176			
U 0803, 2042.95	:6177	T1.3:		: IF NO ERROR UPDATE ERROR NUMBER
U 0804, 3E82.95	:6178	MOV LS[ERROR.NUMBER] TO WR[1]		:
U 0805, 0960.04	:6179	INC WR[1]		:
	:6180	MOV WR[1] TO LS[ERROR.NUMBER]		:
	:6181	JMP [LOOP.T1.3]		: SET ADDRESS FOR NEXT SKIP
U 1600, 10D0.15	:6182	1600:		
U 1601, 90C0.15	:6183	LOOP.T1.3:		: SCOPE SYNC POINT
	:6184	MISC [SET.CP.ACK]		: CLEAR CPU ACK FOR SYNCING
	:6185	MISC [CLR.CP.ATTN.AND.ACK]		: NOP
U 1602, 5B00.1E	:6186	NOP		: SHOULD SKIP NEXT INSTRUCTION
U 1603, 0874.84	:6187	SKIP		: ERROR IF HERE
	:6188	JMP [ERROR.T1.3]		
U 1604, B682.95	:6189			
U 1605, 2042.95	:6190	T1.4:		: IF NO ERROR UPDATE ERROR NUMBER
U 1606, 3E82.95	:6191	MOV LS[ERROR.NUMBER] TO WR[1]		:
U 1607, 8900.44	:6192	INC WR[1]		:
	:6193	MOV WR[1] TO LS[ERROR.NUMBER]		:
	:6194	JMP [LOOP.T1.4]		: SET ADDRESS FOR NEXT SKIP
	:6195			
U 1004, 10D0.15	:6196	1004:		
U 1005, 90C0.15	:6197	LOOP.T1.4:		: SCOPE SYNC POINT
	:6198	MISC [SET.CP.ACK]		: CLEAR CPU ACK FOR SYNCING
	:6199	MISC [CLR.CP.ATTN.AND.ACK]		: NOP
U 1006, 5B00.1E	:6200	NOP		: SHOULD SKIP NEXT INSTRUCTION
U 1007, 8874.C4	:6201	SKIP		: ERROR IF HERE
	:6202	JMP [ERROR.T1.4]		
U 1008, B682.95				
U 1009, 2042.95		T1.5:		: IF NO ERROR UPDATE ERROR NUMBER
U 100A, 3E82.95		MOV LS[ERROR.NUMBER] TO WR[1]		:
U 100B, 0880.C4		INC WR[1]		:
		MOV WR[1] TO LS[ERROR.NUMBER]		:
		JMP [LOOP.T1.5]		

	:6203	80C:		:SET ADDRESS FOR NEXT SKIP
	:6204	LOOP.T1.5:		
U 080C, 10D0,15	:6205	MISC [SET.CP.ACK]		:SCOPE SYNC POINT
U 080D, 90C0,15	:6206	MISC [CLR.CP.ATTN.AND.ACK]		:CLEAR CPU ACK FOR SYNCING
	:6207	NOP		:NOP
U 080E, 5B00,1E	:6208	SKIP		: SHOULD SKIP NEXT INSTRUCTION
U 080F, 0875,04	:6209	JMP [ERROR.T1.5]		:ERROR IF HERE
	:6210			
U 0810, B682,95	:6211	T1.6: MOV LS[ERROR.NUMBER] TO WR[1]		:IF NO ERROR UPDATE ERROR NUMBER
U 0811, 2042,95	:6212	INC WR[1]		:
U 0812, 3E82,95	:6213	MOV WR[1] TO LS[ERROR.NUMBER]		:
U 0813, 8881,C4	:6214	JMP [LOOP.T1.6]		
	:6215			
	:6216	81C:		:SET ADDRESS FOR NEXT SKIP
U 081C, 10D0,15	:6217	LOOP.T1.6:		
U 081D, 90C0,15	:6218	MISC [SET.CP.ACK]		:SCOPE SYNC POINT
	:6219	MISC [CLR.CP.ATTN.AND.ACK]		:CLEAR CPU ACK FOR SYNCING
U 081E, 5B00,1E	:6220	NOP		:NOP
U 081F, 8875,44	:6221	SKIP		: SHOULD SKIP NEXT INSTRUCTION
	:6222	JMP [ERROR.T1.6]		:ERROR IF HERE
U 0820, B682,95	:6223	T1.7: MOV LS[ERROR.NUMBER] TO WR[1]		:IF NO ERROR UPDATE ERROR NUMBER
U 0821, 2042,95	:6224	INC WR[1]		:
U 0822, 3E82,95	:6225	MOV WR[1] TO LS[ERROR.NUMBER]		:
U 0823, 0883,C4	:6226	JMP [LOOP.T1.7]		
	:6227			
	:6228	83C:		:SET ADDRESS FOR NEXT SKIP
U 083C, 10D0,15	:6229	LOOP.T1.7:		
U 083D, 90C0,15	:6230	MISC [SET.CP.ACK]		:SCOPE SYNC POINT
	:6231	MISC [CLR.CP.ATTN.AND.ACK]		:CLEAR CPU ACK FOR SYNCING
U 083E, 5B00,1E	:6232	NOP		:NOP
U 083F, 8875,84	:6233	SKIP		: SHOULD SKIP NEXT INSTRUCTION
	:6234	JMP [ERROR.T1.7]		:ERROR IF HERE
U 0840, B682,95	:6235	T1.8: MOV LS[ERROR.NUMBER] TO WR[1]		:IF NO ERROR UPDATE ERROR NUMBER
U 0841, 2042,95	:6236	INC WR[1]		:
U 0842, 3E82,95	:6237	MOV WR[1] TO LS[ERROR.NUMBER]		:
U 0843, 8887,C4	:6238	JMP [LOOP.T1.8]		
	:6239			
	:6240	87C:		:SET ADDRESS FOR NEXT SKIP
U 087C, 10D0,15	:6241	LOOP.T1.8:		
U 087D, 90C0,15	:6242	MISC [SET.CP.ACK]		:SCOPE SYNC POINT
	:6243	MISC [CLR.CP.ATTN.AND.ACK]		:CLEAR CPU ACK FOR SYNCING
U 087E, 5B00,1E	:6244	NOP		:NOP
U 087F, 0875,C4	:6245	SKIP		: SHOULD SKIP NEXT INSTRUCTION
	:6246	JMP [ERROR.T1.8]		:ERROR IF HERE
U 0880, B682,95	:6247	T1.9: MOV LS[ERROR.NUMBER] TO WR[1]		:IF NO ERROR UPDATE ERROR NUMBER
U 0881, 2042,95	:6248	INC WR[1]		:
U 0882, 3E82,95	:6249	MOV WR[1] TO LS[ERROR.NUMBER]		:
U 0883, 088F,C4	:6250	JMP [LOOP.T1.9]		
	:6251			
	:6252	8FC:		:SET ADDRESS FOR NEXT SKIP
U 08FC, 10D0,15	:6253	LOOP.T1.9:		
U 08FD, 90C0,15	:6254	MISC [SET.CP.ACK]		:SCOPE SYNC POINT
	:6255	MISC [CLR.CP.ATTN.AND.ACK]		:CLEAR CPU ACK FOR SYNCING
U 08FE, 5B00,1E	:6256	NOP		:NOP
U 08FF, 0876,04	:6257	SKIP		: SHOULD SKIP NEXT INSTRUCTION
		JMP [ERROR.T1.9]		:ERROR IF HERE


```

:6258 T1.A:
U 0900, B682,95 :6259 MOV LS[ERROR.NUMBER] TO WR[1] ;IF NO ERROR UPDATE ERROR NUMBER
U 0901, 2042,95 :6260 INC WR[1] ;
U 0902, 3E82,95 :6261 MOV WR[1] TO LS[ERROR.NUMBER] ;
U 0903, 889F,C4 :6262 JMP [LOOP.T1.A] ;
:6263 9FC: ;SET ADDRESS FOR NEXT SKIP
:6264 LOOP.T1.A:
U 09FC, 10D0,15 :6265 MISC [SET.CP.ACK] ;SCOPE SYNC POINT
U 09FD, 90C0,15 :6266 MISC [CLR.CP.ATTN.AND.ACK] ;CLEAR CPU ACK FOR SYNCING
:6267 NOP ;NOP
:6268 SKIP ;SHOULD SKIP NEXT INSTRUCTION
U 09FE, 5800,1E :6269 JMP [ERROR.T1.A] ;ERROR IF HERE
:6270 T1.B:
U 0A00, B682,95 :6271 MOV LS[ERROR.NUMBER] TO WR[1] ;IF NO ERROR UPDATE ERROR NUMBER
U 0A01, 2042,95 :6272 INC WR[1] ;
U 0A02, 3E82,95 :6273 MOV WR[1] TO LS[ERROR.NUMBER] ;
U 0A03, 08BF,C4 :6274 JMP [LOOP.T1.B] ;
:6275 0BFC: ;SET ADDRESS FOR NEXT SKIP
:6276 LOOP.T1.B:
U 0BFC, 10D0,15 :6277 MISC [SET.CP.ACK] ;SCOPE SYNC POINT
U 0BFD, 90C0,15 :6278 MISC [CLR.CP.ATTN.AND.ACK] ;CLEAR CPU ACK FOR SYNCING
:6279 NOP ;NOP
:6280 SKIP ;SHOULD SKIP NEXT INSTRUCTION
U 0BFE, 5800,1E :6281 JMP [ERROR.T1.B] ;ERROR IF HERE
U 0BFF, 8876,84 :6282 JMP [END.T1] ;JUMP OVER ERROR ROUTINES
:6283 740:
:6284 ERROR.T1.1:
U 0740, 10E0,15 :6285 MISC [SET.CP.ATTN] ;REPORT ERROR #1 TO CONSOLE
:6286 WAIT.T1.1:
U 0741, 0874,14 :6287 JMP [WAIT.T1.1] ;LOOP WAITING FOR CONSOLE TO RESPOND
U 0742, 097F,D4 :6288 JMP [LOOP.T1.1] ;LOOP ON ERROR IF ENABLED
U 0743, 0980,14 :6289 JMP [T1.2] ;ELSE NEXT TEST
:6290 ERROR.T1.2:
U 0744, 10E0,15 :6291 MISC [SET.CP.ATTN] ;REPORT ERROR #2 TO CONSOLE
:6292 WAIT.T1.2:
U 0745, 8874,54 :6293 JMP [WAIT.T1.2] ;LOOP WAITING FOR CONSOLE TO RESPOND
U 0746, 887F,E4 :6294 JMP [LOOP.T1.2] ;LOOP ON ERROR IF ENABLED
U 0747, 8880,24 :6295 JMP [T1.3] ;ELSE NEXT TEST
:6296 ERROR.T1.3:
U 0748, 10E0,15 :6297 MISC [SET.CP.ATTN] ;REPORT ERROR #3 TO CONSOLE
:6298 WAIT.T1.3:
U 0749, 8874,94 :6299 JMP [WAIT.T1.3] ;LOOP WAITING FOR CONSOLE TO RESPOND
U 074A, 0960,04 :6300 JMP [LOOP.T1.3] ;LOOP ON ERROR IF ENABLED
U 074B, 8960,44 :6301 JMP [T1.4] ;ELSE NEXT TEST
:6302 ERROR.T1.4:
U 074C, 10E0,15 :6303 MISC [SET.CP.ATTN] ;REPORT ERROR #4 TO CONSOLE
:6304 WAIT.T1.4:
U 074D, 0874,D4 :6305 JMP [WAIT.T1.4] ;LOOP WAITING FOR CONSOLE TO RESPOND
U 074E, 8900,44 :6306 JMP [LOOP.T1.4] ;LOOP ON ERROR IF ENABLED
U 074F, 8900,84 :6307 JMP [T1.5] ;ELSE NEXT TEST
:6308 ERROR.T1.5:
U 0750, 10E0,15 :6309 MISC [SET.CP.ATTN] ;REPORT ERROR #5 TO CONSOLE
:6310 WAIT.T1.5:
U 0751, 8875,14 :6311 JMP [WAIT.T1.5] ;LOOP WAITING FOR CONSOLE TO RESPOND
U 0752, 0880,C4 :6312 JMP [LOOP.T1.5] ;LOOP ON ERROR IF ENABLED

```


U 0753, 8881,04	:6313	JMP [T1.6]	:ELSE NEXT TEST
U 0754, 10E0,15	:6314	ERROR.T1.6:	
	:6315	MISC [SET.CP.ATTN]	:REPORT ERROR #6 TO CONSOLE
	:6316	WAIT.T1.6:	
U 0755, 0875,54	:6317	JMP [WAIT.T1.6]	:LOOP WAITING FOR CONSOLE TO RESPOND
U 0756, 8881,C4	:6318	JMP [LOOP.T1.6]	:LOOP ON ERROR IF ENABLED
U 0757, 8882,04	:6319	JMP [T1.7]	:ELSE NEXT TEST
	:6320	ERROR.T1.7:	
U 0758, 10E0,15	:6321	MISC [SET.CP.ATTN]	:REPORT ERROR #7 TO CONSOLE
	:6322	WAIT.T1.7:	
U 0759, 0875,94	:6323	JMP [WAIT.T1.7]	:LOOP WAITING FOR CONSOLE TO RESPOND
U 075A, 0883,C4	:6324	JMP [LOOP.T1.7]	:LOOP ON ERROR IF ENABLED
U 075B, 8884,04	:6325	JMP [T1.8]	:ELSE NEXT TEST
	:6326	ERROR.T1.8:	
U 075C, 10E0,15	:6327	MISC [SET.CP.ATTN]	:REPORT ERROR #8 TO CONSOLE
	:6328	WAIT.T1.8:	
U 075D, 8875,D4	:6329	JMP [WAIT.T1.8]	:LOOP WAITING FOR CONSOLE TO RESPOND
U 075E, 8887,C4	:6330	JMP [LOOP.T1.8]	:LOOP ON ERROR IF ENABLED
U 075F, 8888,04	:6331	JMP [T1.9]	:ELSE NEXT TEST
	:6332	ERROR.T1.9:	
U 0760, 10E0,15	:6333	MISC [SET.CP.ATTN]	:REPORT ERROR #9 TO CONSOLE
	:6334	WAIT.T1.9:	
U 0761, 8876,14	:6335	JMP [WAIT.T1.9]	:LOOP WAITING FOR CONSOLE TO RESPOND
U 0762, 088F,C4	:6336	JMP [LOOP.T1.9]	:LOOP ON ERROR IF ENABLED
U 0763, 8890,04	:6337	JMP [T1.A]	:ELSE NEXT TEST
	:6338	ERROR.T1.A:	
U 0764, 10E0,15	:6339	MISC [SET.CP.ATTN]	:REPORT ERROR #A TO CONSOLE
	:6340	WAIT.T1.A:	
U 0765, 0876,54	:6341	JMP [WAIT.T1.A]	:LOOP WAITING FOR CONSOLE TO RESPOND
U 0766, 889F,C4	:6342	JMP [LOOP.T1.A]	:LOOP ON ERROR IF ENABLED
U 0767, 88A0,04	:6343	JMP [T1.B]	:ELSE NEXT TEST
	:6344	ERROR.T1.B:	
U 0768, 10E0,15	:6345	MISC [SET.CP.ATTN]	:REPORT ERROR #B TO CONSOLE
	:6346	WAIT.T1.B:	
U 0769, 0876,94	:6347	JMP [WAIT.T1.B]	:LOOP WAITING FOR CONSOLE TO RESPOND
U 076A, 08BF,C4	:6348	JMP [LOOP.T1.B]	:LOOP ON ERROR IF ENABLED
	:6349	END.T1:	

:6350 Page "TEST 2 - Skip, Jump On ALU Z Test (M8390 CPU Module)"

:6351 ++

:6352

:6353

:6354

:6355

:6356

:6357

:6358

:6359

:6360

:6361

:6362

:6363

:6364

:6365

:6366

:6367

:6368

:6369

:6370

:6371

:6372

:6373

:6374

:6375

:6376

:6377

:6378

:6379

:6380

:6381

:6382

:6383

:6384

:6385

:6386

:6387

:6388

:6389

:6390

:6391

:6392

:6393

:6394

:6395

:6396

:6397

:6398

:6399

:6400

:6401

:6402

:6403

:6404

Test Description:

This test checks the branch control logic for branching on ALU=0 and ALU not equal 0. This branch will be used in the error routine to check test results but the error routine will not be used until logic used in the error routine has been tested. The data path used is as follows: SCTL bits 0-2 select ALU Z H as the input to the 8 to 1 MUX which feeds the SEQ.CTL PAL. SCTL bits 0-4 set up the PAL to be controlled by this input. The PAL will output to the 3 input negated OR gate which controls INCR CIN H. This output goes to the adder which controls whether BUS uPC is incremented or not. The result is sent through the 2 to 1 MUX resulting in NAD 00 L through NAD 03 L. For the Jump on ALU Z, the data path is as follows: JCTL bits 0-2 select ALU Z as the input to the 8 to 1 MUX. JCTL bits 0-3 set up the SEQ.CTL PAL to be controlled by the output from the 8 to 1 MUX. The PAL will control the micro sequencer logic with the two ENABLE JUMP outputs. If the ENABLE JUMP outputs are asserted, CSR bits 4-17 will determine the next address, if the outputs are not asserted, the next sequential address will be executed.

Assumptions:

It is assumed that the previous test (unconditional skip) has run successfully and that the ALU Z bit has been previously tested by the console. One other assumption is that the MUX&INCR CTL PAL will not cause an unexpected skip.

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts).
- 2) Issue a CLR WR[0] instruction to put zeros in the 2901 WR[0] SCTL=skip if ALU=0.
- 3) Report error if no skip occurred, else INC 2901 to put a 1 into the low bit of 2901. SCTL=skip if ALU=0.
- 4) This time no skip should occur. Report error if skip, else shift 1 bit to next bit position (left shift) filling low bits with 0's. SCTL=skip if ALU=0.
- 5) Repeat step 4 until a 1 has been shifted across all bit positions (total of 32 shifts).
- 6) Repeat steps 2 and 3, this time with SCTL=skip if ALU not equal 0, checking for no skip and skip respectively.
- 7) Issue a CLR WR[0] instruction to put zeros in the 2901 WR[0] JCTL=jump if ALU=0.
- 8) Report error if no jump occurred, else INC 2901 to put a 1 into the low bit of 2901. JCTL=jump if ALU=0.
- 9) This time no jump should occur. Report error if jump.
- 10) Repeat steps 7 and 8, this time with JCTL=jump if ALU not equal 0, checking for no jump and jump respectively.

Errors:

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

J 12
TEST 2 - Skip, Jump 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 2 - Skip, Jump On ALU Z Test (M8390 CPU Module)

Fiche 1 Frame J12

Sequence 152
Rev 01.00 Page 146

:6405
:6406
:6407
:6408
:6409
:6410
:6411
:6412
:6413
:6414
:6415
:6416
:6417
:6418
:6419
:6420
:6421
:6422
:6423
:6424
:6425
:6426
:6427
:6428
:6429
:6430
:6431
:6432
:6433
:6434
:6435
:6436
:6437
:6438
:6439
:6440
:6441
:6442
:6443
:6444
:6445
:6446
:6447
:6448
:6449
:6450
:6451
:6452
:6453
:6454
:6455
:6456
:6457
:6458
:6459

Error 1 - Skip on ALU=0 failed to skip when ALU was cleared.
Error 2 - Skip on ALU=0 skipped when ALU was not equal to zero.
Error 3 - Skip on ALU not equal 0 skipped when ALU was cleared.
Error 4 - Skip on ALU not equal 0 failed to skip when ALU was not equal to zero.
Error 5 - Jump on ALU=0 failed to jump when ALU was cleared.
Error 6 - Jump on ALU=0 jumped when ALU was not equal to zero.
Error 7 - Jump on ALU not equal 0 jumped when ALU was cleared.
Error 8 - Jump on ALU not equal 0 failed to jump when ALU was not equal to zero.

Debug:

Error 1 or 2 - This error indicates a problem with the 8 to 1 mux which outputs to the SEQ.CTL PAL, the mux inputs, or the SEQ.CTL PAL itself. When the instruction with SCTL=skip if ALU=0 is executed, CSR bits 2-0 at the mux should equal 1(H) and ALU Z H should be high. The output on pin 6 should be low going to the SEQ.CTL PAL. This PAL should have CSR bits 4-0 equal to 9(H), and JUMP INSTR L high. If these inputs are good, and pin 3 is low, then the problem is with this PAL. The logic after the PAL has been checked in a previous test.

Error 3 or 4 - This error, if not proceeded by error 1 or 2, indicates a problem with the SEQ.CTL PAL or it's CSR inputs. CSR bits 4-0 should equal 1. If this is correct the problem is probably in the PAL itself. All other logic involved has been checked previously.

Error 5 or 6 - If this is the first error, suspect the SEQ.CTL PAL. The inputs to this PAL should be as follows: CSR bits 3-0 equal 9(H), JUMP INSTR L should be low, and CSR 19 should be high. The ENABLE JUMP LO L and ENABLE JUMP L outputs should be low for error 4 and high for error 5. The output on pin 19 should be high in both cases. These inputs and outputs can be checked during the JUMP instruction. The rest of the jump logic has been tested previously.

Error 7 or 8 - Same as error 5 or 6 except CSR bits 3-0 should equal 1(H).

NOTE: SINCE SKIP ON ALU=0 IS BEING TESTED IN THIS TEST, IT IS NOT POSSIBLE TO USE A LOOP ON THE SHIFTING PATTERN, SINCE THE ONLY WAY TO EXIT THE LOOP WOULD BE TO USE THE SKIP. THEREFORE THIS TEST USES A CONSOLE ROUTINE TO LOOP. HOWEVER, IT IS NOT POSSIBLE TO LOOP ON ERROR IF THE ERROR GOES AWAY SINCE WE HAVE NO WAY TO COMPARE THE LAST ERROR WITH THE CURRENT ERROR.

T.2:

U 076B, B65E,15
U 076C, 3E80,15
U 076D, 10E0,15

MOV LS[BEGIN.TEST] TO WR[0]
MOV WR[0] TO LS[CONTROL.STATUS]

MISC [SET.CP.ATTN]

WAIT.T2.0:

:SET BIT 15 IN WR[0] FOR CONTROL AND
:STATUS WORD
:SET BIT 15 IN CONTROL AND STATUS WORD
:BIT 15 INDICATES START OF TEST TO
:CONSOLE
:ISSUE CPU ATTN TO CONSOLE


```

U 076E, 8876,E4 :6460      JMP [WAIT.T2.0]      ;LOOP HERE WAITING FOR CONSOLE TO
:6461                      ;RESPOND
U 076F, 2F80,15 :6462      CLR WR[0]      ;PUT A 0 IN WR 0
U 0770, 3E8A,15 :6463      MOV WR[0] TO LS[ERROR.MASK] ;CLEAR ERROR MASK. MASK IS NOT USED IN
:6464                      ;THIS TEST SINCE WE ARE CHECKING SKIPS
U 0771, 2040,15 :6465      INC WR[0]      ;MAKE WR[0] = 1
U 0772, BE82,15 :6466      MOV WR[0] TO LS[ERROR.NUMBER] ;SET UP ERROR NUMBER TO FIRST ERROR
U 0773, 2040,15 :6467      INC WR[0]      ;SET WRO TO 2
U 0774, 3E8C,15 :6468      MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
U 0775, B64A,15 :6469      MOV LS[#20] TO WR[0] ;SET UP FOR LOOP COUNT
U 0776, BE8E,15 :6470      MOV WR[0] TO LS[LOOP.CNT] ;SET UP LOOP COUNT IN LS FOR 32
:6471
:6472
:6473
:6474
U 0777, 10D0,15 :6475      LOOP.T2.1:      MISC [SET.CP.ACK]      ;SCOPE SYNC POINT
U 0778, 90C0,15 :6476      MISC [CLR.CP.ATTN.AND.ACK] ;CLEAR SIGNALS FOR SYNCING
:6477      CLR WR[0]      ;CLEAR WR[0] FOR TEST
U 0779, AF80,35 :6478      DT(LONG)&SET.ALU.CC ;SET CONDITION CODES
U 077A, 5B00,09 :6479      SKIP.IF[EQ] ;ALU=0 SO THIS SHOULD SKIP
U 077B, 087C,04 :6480      JMP [ERROR.T2.1] ;REPORT ERROR IF NO SKIP
:6481
U 077C, B642,95 :6482      T2.2:      MOV LS[#2] TO WR[1] ;LOAD ERROR NUMBER 2 IN WR[1]
U 077D, 3E82,95 :6483      MOV WR[1] TO LS[ERROR.NUMBER] ;UPDATE ERROR NUMBER IN LS
:6484
:6485      INC WR[0]      ;PUT A ONE IN WR[0]
:6486      DT(LONG)&SET.ALU.CC ;SET CONDITION CODES
U 077E, A040,35 :6487      LOOP.T2.2:      MISC [SET.CP.ACK]      ;SCOPE SYNC POINT
U 077F, 10D0,15 :6488      MISC [CLR.CP.ATTN.AND.ACK] ;CLEAR SIGNALS FOR SYNCING
U 0780, 90C0,15 :6489      SKIP.IF[EQ] ;ALU NOT = 0 SO NO SKIP SHOULD OCCUR
U 0781, 5B00,09 :6490      JMP [T2.2A] ;JUMP TO NEXT DATA PATTERN (LAST
:6491                      ;PATTERN PASSED)
U 0783, B63C,95 :6492      MOV LS[ERR.CON.NA] TO WR[1] ;ERROR IF HERE
:6493                      ;LOAD WR[1] WITH A CONSTANT FROM LS
:6494                      ;THAT SETS BITS 8,10, AND 23
:6495                      ;LOAD CONTROL AND STATUS WORD. BIT
U 0784, BE80,95 :6496      MOV WR[1] TO LS[CONTROL.STATUS] ;8 SET INDICATES AN ERROR, BIT 10
:6497                      ;SET INDICATES CONSOLE CONTROLS LOOP
:6498                      ;ON ERROR, BIT 23 SET INDICATES THAT
:6499                      ;CONSOLE SHOULD NOT PRINT EXPECTED OR
:6500                      ;RECEIVED DATA ON ERROR REPORT.
:6501                      ;ERROR. SEND CPU AT.N TO CONSOLE TO REPORT
U 0785, 10E0,15 :6502      WAIT.T2.2:      MISC [SET.CP.ATTN] ;WAIT FOR CONSOLE TO RESPOND
:6503      JMP [WAIT.T2.2] ;LOOP ON ERROR IF ENABLED
U 0786, 8878,64 :6504      JMP [LOOP.T2.2]
U 0787, 8877,F4 :6505
:6506
:6507
:6508
U 0788, A340,35 :6509      T2.2A:      ROR WR[0]      ;NEXT DATA PATTERN
U 0789, B672,95 :6510      DT(LONG)&SET.ALU.CC ;SET CONDITION CODES
:6511      MOV LS[BIT25] TO WR[1] ;LOAD WR[1] WITH CONSTANT FROM LS THAT
:6512                      ;SETS BIT 25
U 078A, BE80,95 :6513      MOV WR[1] TO LS[CONTROL.STATUS] ;LOAD CONTROL AND STATUS WORD WITH BIT
:6514                      ;25 SET TO INDICATE THAT CONSOLE IS TO
:6515                      ;PERFORM LOOPING

```

U 078B, 10E0,15	:6515	MISC [SET.CP.ATTN]	:CALL CONSOLE
U 078C, 8878,C4	:6516	WAIT.T2.2A:	
U 078D, 8877,F4	:6517	JMP [WAIT.T2.2A]	:LOOP SELF UNTIL CONSOLE RESPONDS
	:6518	JMP [LOOP.T2.2]	:CONSOLE RETURNS HERE IF LOOP COUNT NOT
	:6519		: EXHAUSTED, ELSE GOES TO NEXT WORD
	:6520		
U 078E, B63C,95	:6521	ALU.NOT.EQUAL:	
U 078F, BE80,95	:6522	MOV LS[ERR.CON.NA] TO WR[1]	:LOAD WR[1] WITH A CONSTANT FROM LS
	:6523		: THAT SETS BITS 8,10, AND 23
	:6524	MOV WR[1] TO LS[CONTROL.STATUS]	:LOAD CONTROL AND STATUS WORD. BIT
	:6525		: 8 SET INDICATES AN ERROR, BIT 10
	:6526		: SET INDICATES CONSOLE CONTROLS LOJP
	:6527		: ON ERROR, BIT 23 SET INDICATES THAT
	:6528		: CONSOLE SHOULD NOT PRINT EXPECTED OR
	:6529		: RECEIVED DATA ON ERROR REPORT. THIS
	:6530		: WORD IS LOADED NOW SO THAT IT NEEDN'T
	:6531		: BE LOADED AGAIN IF AN ERROR OCCURS.
U 0790, B6C0,95	:6532	MOV LS[#3(H)] TO WR[1]	:SET UP WR[1] FOR ERROR NUMBER
U 0791, 3E82,95	:6533	MOV WR[1] TO LS[ERROR.NUMBER]	:UPDATE ERROR NUMBER TO ERROR 3
	:6534		
U 0792, 10D0,15	:6535	LOOP.T2.3:	
U 0793, 90C0,15	:6536	MISC [SET.CP.ACK]	:SCOPE SYNC POINT
	:6537	MISC [CLR.CP.ATTN.AND.ACK]	:CLEAR SIGNALS FOR SYNCING
	:6538	CLR WR[0],	:CLEAR WR [0] AGAIN.
U 0794, AF80,35	:6539	DT(LONG)&SET.ALU.CC	: SET CONDITION CODES
U 0795, DB00,01	:6540	SKIP.IF[NEQ]	:ALU=0 SO THIS SHOULD NOT SKIP
U 0796, 0879,A4	:6541	JMP [T2.4]	:THIS PASSED. GO TO NEXT PART
	:6542		
U 0797, 10E0,15	:6543	MISC [SET.CP.ATTN]	:ERROR HERE. CALL THE CONSOLE TO REPORT
	:6544	WAIT.T2.3:	
U 0798, 8879,84	:6545	JMP [WAIT.T2.3]	:LOOP WAITING FOR CONSOLE TO RESPOND
U 0799, 8879,24	:6546	JMP [LOOP.T2.3]	:LOOP ON ERROR IF ENABLED
	:6547		
	:6548		
U 079A, 2042,95	:6549	T2.4:	
U 079B, 3E82,95	:6550	INC WR[1]	:INCREMENT ERROR NUMBER
	:6551	MOV WR[1] TO LS[ERROR.NUMBER]	:AND PUT NEW ERROR NUMBER OF 4 IN LS
	:6552		
U 079C, A040,35	:6553	INC WR[0],	:WR[0] NOW NON-ZERO
	:6554	DT(LONG)&SET.ALU.CC	: SET CONDITION CODES
U 079D, 10D0,15	:6555	LOOP.T2.4:	
U 079E, 90C0,15	:6556	MISC [SET.CP.ACK]	:SCOPE SYNC POINT
U 079F, DB00,01	:6557	MISC [CLR.CP.ATTN.AND.ACK]	:CLEAR SIGNALS FOR SYNCING
U 07A0, 087C,64	:6558	SKIP.IF[NEQ]	:THIS SHOULD SKIP
	:6559	JMP [ERROR.T2.4]	:ERROR IF HERE, GO TO REPORT IT
U 07A1, 2042,95	:6560		
U 07A2, 3E82,95	:6561	T2.5:	
	:6562	INC WR[1]	:INCREMENT ERROR NUMBER
	:6563	MOV WR[1] TO LS[ERROR.NUMBER]	:AND PUT ERROR NUMBER 5 IN LS
U 07A3, 10D0,15	:6564	LOOP.T2.5:	
U 07A4, 90C0,15	:6565	MISC [SET.CP.ACK]	:SCOPE SYNC POINT
	:6566	MISC [CLR.CP.ATTN.AND.ACK]	:CLEAR SIGNALS FOR SYNCING
U 07A5, AF80,35	:6567	CLR WR[0],	:ALU=0 NOW
U 07A6, 887A,A9	:6568	DT(LONG)&SET.ALU.CC	: SET CONDITION CODES
	:6569	JMP.IF[EQL] TO [T2.6]	:THIS SHOULD JUMP TO NEXT PART

ZZ-ENKCA-1.0 : ENKCA.MIC
 : ENKCA.MCR
 : ENKCA.MIC

MICRO2 1L(03) TEST 2 - Skip, Jump 3-MAR-82 09:34:33
 TEST 2 - Skip, Jump On ALU Z Test (M8390 CPU Module)

M 12
 3-MAR-82
 ENKCA Micro-diagnostic for DAP module
 Rev 01.00

Fiche 1 Frame M12
 Sequence 155
 Page 149

```

U 07A7, 10E0,15 :6570 MISC [SET.CP.ATTN] ;ERROR IF HERE. REPORT TO CONSOLE
:6571 WAIT.T2.5:
U 07A8, 887A,84 :6572 JMP [WAIT.T2.5] ;LOOP WAITING FOR CONSOLE
U 07A9, 087A,34 :6573 JMP [LOOP.T2.5] ;ERROR LOOP IF ENABLED
:6574
:6575 T2.6:
U 07AA, 2042,95 :6576 INC WR[1] ;INCREMENT ERROR NUMBER
U 07AB, 3E82,95 :6577 MOV WR[1] TO LS[ERROR.NUMBER] ;PUT ERROR NUMBER OF 6 IN LS
:6578
:6579 INC WR[0] ;ALU NOT=0
U 07AC, A040,35 :6580 DT(LONG)&SET.ALU.CC ; SET CONDITION CODES
:6581
:6582 LOOP.T2.6:
U 07AD, 10D0,15 :6583 MISC [SET.CP.ACK] ;SCOPE SYNC POINT
U 07AE, 90C0,15 :6584 MISC [CLR.CP.ATTN.AND.ACK] ;CLEAR SIGNALS FOR SYNCING
U 07AF, 887C,A9 :6585 JMP.IF[EQL] TO [ERROR.T2.6] ;THIS SHOULD NOT JUMP. REPORT ERROR
:6586 ; IF JUMP OCCURS
:6587
:6588 T2.7:
U 07B0, 2042,95 :6589 INC WR[1] ;INCREMENT ERROR NUMBER
U 07B1, 3E82,95 :6590 MOV WR[1] TO LS[ERROR.NUMBER] ;PUT ERROR NUMBER OF 7 IN LS
:6591
:6592 LOOP.T2.7:
U 07B2, 10D0,15 :6593 MISC [SET.CP.ACK] ;SCOPE SYNC POINT
U 07B3, 90C0,15 :6594 MISC [CLR.CP.ATTN.AND.ACK] ;CLEAR SIGNALS FOR SYNCING
:6595 CLR WR[0] ;ALU=0
:6596 DT(LONG)&SET.ALU.CC ; SET CONDITION CODES
U 07B4, AF80,35 :6597 JMP.IF[NEQ] TO [ERROR.T2.7] ;THIS SHOULD NOT JUMP. REPORT ERROR IS
U 07B5, 887C,E1 :6598 ; JUMP OCCURS
:6599
:6600 T2.8:
U 07B6, 2042,95 :6601 INC WR[1] ;INCREMENT ERROR NUMBER
U 07B7, 3E82,95 :6602 MOV WR[1] TO LS[ERROR.NUMBER] ;PUT ERROR NUMBER OF 8 IN LS
:6603
:6604 INC WR[0] ;ALU NOT = 0
U 07B8, A040,35 :6605 DT(LONG)&SET.ALU.CC ; SET CONDITION CODES
:6606
:6607 LOOP.T2.8:
U 07B9, 10D0,15 :6608 MISC [SET.CP.ACK] ;SCOPE SYNC POINT
U 07BA, 90C0,15 :6609 MISC [CLR.CP.ATTN.AND.ACK] ;CLEAR SIGNALS FOR SYNCING
U 07BB, 087D,21 :6610 JMP.IF[NEQ] TO [END.T2] ;THIS SHOULD JUMP TO END OF TEST
:6611
:6612 MISC [SET.CP.ATTN] ;ERROR IF HERE. INFORM CONSOLE OF ERROR
U 07BC, 10E0,15 :6613 WAIT.T2.8:
:6614 JMP [WAIT.T2.8] ;LOOP WAITING FOR CONSOLE TO RESPOND
U 07BD, 087B,D4 :6615 JMP [LOOP.T2.8] ;LOOP ON ERROR IF ENABLED
U 07BE, 887B,94 :6616 JMP [END.T2] ;JUMP OVER ERROR REPORTS
U 07BF, 087D,24 :6617
:6618
:6619 ERROR.T2.1:
U 07C0, B63C,95 :6620 MOV LS[ERR.CON.NA] TO WR[1] ;ERROR IF HERE
:6621 ; LOAD WR[1] WITH A CONSTANT FROM LS
:6622 ; THAT SETS BITS 8,10, AND 23
U 07C1, BE80,95 :6623 MOV WR[1] TO LS[CONTROL.STATUS] ;LOAD CONTROL AND STATUS WORD. BIT
:6624 ; 8 SET INDICATES AN ERROR, BIT 10
: ; SET INDICATES CONSOLE CONTROLS LOOP
: ; ON ERROR, BIT 23 SET INDICATES THAT
: ; CONSOLE SHOULD NOT PRINT EXPECTED OR
: ; RECEIVED DATA ON ERROR REPORT.
  
```


U 07C2, 10E0,15	:6625	MISC [SET.CP.ATTN]	;ERROR NUMBER 1. REPORT TO CONSOLE
	:6626	WAIT.T2.1:	
U 07C3, 087C,34	:6627	JMP [WAIT.T2.1]	;LOOP WAITING FOR CONSOLE TO RESPOND
U 07C4, 0877,74	:6628	JMP [LOOP.T2.1]	;LOOP ON ERROR IF ENABLED
U 07C5, 8877,C4	:6629	JMP [T2.2]	;ELSE NEXT PART OF THIS TEST
	:6630		
	:6631	ERROR.T2.4:	
U 07C6, 10E0,15	:6632	MISC [SET.CP.ATTN]	;ERROR NUMBER 4. REPORT TO CONSOLE
	:6633	WAIT.T2.4:	
U 07C7, 887C,74	:6634	JMP [WAIT.T2.4]	;LOOP WAITING FOR CONSOLE TO RESPOND
U 07C8, 8879,D4	:6635	JMP [LOOP.T2.4]	;LOOP ON ERROR IF ENABLED
U 07C9, 887A,14	:6636	JMP [T2.5]	;ELSE NEXT PART OF THIS TEST
	:6637		
	:6638	ERROR.T2.6:	
U 07CA, 10E0,15	:6639	MISC [SET.CP.ATTN]	;ERROR NUMBER 6. REPORT TO CONSOLE
	:6640	WAIT.T2.6:	
U 07CB, 887C,B4	:6641	JMP [WAIT.T2.6]	;LOOP WAITING FOR CONSOLE TO RESPOND
U 07CC, 887A,D4	:6642	JMP [LOOP.T2.6]	;LOOP ON ERROR IF ENABLED
U 07CD, 887B,04	:6643	JMP [T2.7]	;ELSE NEXT PART OF THIS TEST
	:6644		
	:6645	ERROR.T2.7:	
U 07CE, 10E0,15	:6646	MISC [SET.CP.ATTN]	;ERROR NUMBER 7. REPORT TO CONSOLE
	:6647	WAIT.T2.7:	
U 07CF, 087C,F4	:6648	JMP [WAIT.T2.7]	;LOOP WAITING FOR CONSOLE TO RESPOND
U 07D0, 087B,24	:6649	JMP [LOOP.T2.7]	;LOOP ON ERROR IF ENABLED
U 07D1, 887B,64	:6650	JMP [T2.8]	;ELSE NEXT PART OF THIS TEST
	:6651	END.T2:	

Page "TEST 3 - JSR and Stack test (including return and return+1) (M8390 CPU Module)"

++

Test Description:

This test checks the JSR instruction, one location of the stack, and the return and return+1 functions. The data path is as follows: A JUMP with JCTL=JSR will force a jump to the desired destination and in addition will save the calling u-PC on the stack by asserting PUSH STACK and ENABLE SP before the jump is performed. These signals are generated by the SEQ.CTL PAL. ENABLE SP goes to the STACK POINTER PAL which is responsible for providing the stack address and decrementing it before the push. The CSR 03 H input determines whether an increment or decrement is to be performed. On a return, the STACK POINTER PAL is responsible for incrementing the stack address after a pop. An instruction with SCTL=RETURN will cause the SEQ.CTL PAL to assert ENABLE SP and ENABLE RTS. ENABLE SP goes to the STACK POINTER PAL and causes the address to increment after the pop. ENABLE RTS goes to the stack to enable the return u-address on the NAD bus. At the same time, ENABLE RTS disables the output of the normal uPC PALs (by deasserting ENABLE PC L).

Assumptions:

It is assumed that the INCR.HI pal is functioning correctly and that skips on the ALU Z bit have been tested and are known to be good. One other assumption is that the MUX&INCR CTL PAL will not cause an unexpected skip. The console has tested the JSR and RETURN previously but this is the first stack test run at speed. The RETURN+1 has not been previously tested.

NOTE: A failure in the stack on a return may easily send the micro-code into never-never land. Single stepping is the logical method available for tracing this type of failure.

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Clear WR[0]. This WR will be used to determine if a jump occurred.
- 3) Issue a JUMP with JCTL=JSR. This should cause the current uPC to be pushed onto the stack and a normal jump to be taken.
- 4) At the jump destination, increment WR[0] to indicate that a jump occurred. SCTL will be set up to RETURN.
- 5) Report a RETURN error if the instruction after the return is executed.
- 6) If no RETURN error, then check that WR[0] is not 0. If it is, report JSR jump error.
- 7) Issue another JUMP with JCTL=JSR, using an address that will shift a 1 across the stack u-address line outputs.
- 8) Execute a return at the JSR destination. If no error repeat step 7 until all stack outputs have been tested.
- 9) Issue a JUMP with JCTL=JSR, this time to test RTN+1.
- 10) At the jump destination, issue a NOP with SCTL=RTN+1. Report a RTN+1 didn't return error if the next instruction is executed.

:6707 : 11) Test complete if RTN+1 returned to the calling uPC+2, else report
:6708 : RTN+1 failed to increment return address.
:6709 :
:6710 : Errors:
:6711 :
:6712 : Error 1 - SCTL=RETURN failed to return. Fell through to next instruc-
:6713 : tion after RETURN instruction.
:6714 : Error 2 - JCTL=JSR failed to jump to destination.
:6715 : Error 3 - Stack failure on return; bit 13
:6716 : Error 4 - Stack failure on return; bit 12
:6717 : Error 5 - Stack failure on return; bit 11
:6718 : Error 6 - Stack failure on return; bit 10
:6719 : Error 7 - Stack failure on return; bit 9
:6720 : Error 8 - Stack failure on return; bit 8
:6721 : Error 9 - Stack failure on return; bit 7
:6722 : Error A - Stack failure on return; bit 6
:6723 : Error B - Stack failure on return; bit 5
:6724 : Error C - Stack failure on return; bit 4
:6725 : Error D - Stack failure on return; bit 3
:6726 : Error E - Stack failure on return; bit 2
:6727 : Error F - Stack failure on return; bit 1
:6728 : Error 10 - Stack failure on return; bit 0
:6729 : Error 11 - SCTL=RTN+1 failed to return. Fell through to next instruc-
:6730 : tion after RTN+1 instruction.
:6731 : Error 12 - RTN+1 failed to increment return address before returning.
:6732 : It executed a normal return instead.
:6733 :
:6734 : Debug:
:6735 :
:6736 : Note: Errors 1 through 10 indicate a failure on logic previously tested
:6737 : by the console. A failure here probably indicates a timing problem.
:6738 : However, if the machine has been powered down after running the console
:6739 : resident tests, the stack address may be different than that previously
:6740 : tested.
:6741 : Errors 11 and 12 have not been tested previously.
:6742 :
:6743 : Error 1 - This error indicates that the JSR performed a jump correctly
:6744 : but failed to execute a return correctly. The most likely problem is
:6745 : with the signal ENABLE RTS L not being asserted. This signal should
:6746 : be low during the RETURN instruction, forced by pin 17 of the SEQ.CTL
:6747 : PAL and JUMP INSTR L both being high on the input of the NAND gate.
:6748 : The CSR 4-0 inputs to the SEQ.CTL PAL on the return instruction
:6749 : should be 14(H). If ENABLE RTS L is ok, then check that it gets to
:6750 : the stack RAMs correctly. If this is ok, the problem may be with the
:6751 : stack RAMs themselves.
:6752 :
:6753 : Error 2 - This error indicates that the JSR failed to jump. The only
:6754 : difference between a normal JUMP and a JSR is the JCTL value. Check
:6755 : CSR 4-0 going into the SEQ.CTL PAL for a C(H) on the instruction with
:6756 : JCTL=JSR. If these bits are correct, then the SEQ.CTL PAL is probably
:6757 : bad.
:6758 :
:6759 : Error 3-10 - This error indicates a bad stack output bit. It comes up
:6760 : when the address bit being tested fails to go from a zero to a one
:6761 : but all the other bits are good. The error number tells which bit was

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 3 - JSR and St 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 3 - JSR and Stack test (including return and return+1) (M8390 CPU module)

Fiche 1 Frame D13

Sequence 159
Rev 01.00 Page 153

```
:6762      being tested from the list of errors above. Suspect a bad STACK RAM
:6763      if this error occurs.
:6764
:6765      Error 11 - This error indicates that JCTL=RTN+1 failed to execute a
:6766      return but instead fell through to the next instruction. The most
:6767      likely cause for this error is in the SEQ.CTL PAL. The only difference
:6768      between return and return+1 going to this PAL is in CSR 4-0. CSR 4-0
:6769      should be 16(H). If this is correct, suspect the SEQ.CTL PAL.
:6770
:6771      Error 12 - This error indicates that the increment logic used to per-
:6772      form a RTN+1 failed. First check that CSR 4-0 going into the SEQ.CTL
:6773      PAL at the time of the RTN+1 instruction is 16(H). If this is OK, then
:6774      suspect the SEQ.CTL PAL (pin 19 should have gone low when CSR 4-0=16
:6775      (H). The logic after this point has been previously tested).
:6776
:6777
:6778      T.3:
:6779
U 07D2, B65E,15 :6780      MOV LS[BEGIN.TEST] TO WR[0]      ;SET BIT 15 IN WRO FOR CONTROL AND STATUS WORD
U 07D3, 3E80,15 :6781      MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CNTROL AND STATUS WORD
:6782
U 07D4, 10E0,15 :6783      MISC [SET.CP.ATTN]              ;BIT 15 INDICATES START OF TEST TO CONSOLE
:6784      WAIT.T3.0:                      ;ISSUE CPU ATTN TO CONSOLE
U 07D5, 837D,54 :6785      JMP [WAIT.T3.0]                  ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 07D6, 2F80,15 :6786      CLR WR[0]                      ;SET WRO TO 0
U 07D7, 3E8A,15 :6787      MOV WR[0] TO LS[ERROR.MASK]      ;CLEAR ERROR MASK. MASK IS NOT USED IN
:6788                                      ;THIS TEST SINCE WE ARE CHECKING SKIPS
U 07D8, 2040,15 :6789      INC WR[0]                      ;SET WRO TO 1
U 07D9, BE82,15 :6790      MOV WR[0] TO LS[ERROR.NUMBER]    ;SET ERROR NUMBER TO FIRST ERROR
U 07DA, 2040,15 :6791      INC WR[0]                      ;SET WRO TO 2
U 07DB, 3E8C,15 :6792      MOV WR[0] TO LS[MODULE.NUM]      ;SET UP MODULE CODE FOR CPU MODULE
U 07DC, 2F80,15 :6793      CLR WR[0]                      ;SET WRO TO 0
U 07DD, BEA0,15 :6794      MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
U 07DE, 363C,15 :6795      MOV LS[ERR.CON.NA] TO WR[0]    ;LOAD WRO WITH A CONSTANT THAT SETS BITS 8,10, & 23
U 07DF, 3E80,15 :6796      MOV WR[0] TO LS[CONTROL.STATUS] ;LOAD CONTROL AND STATUS WORD. BIT 8
:6797                                      ;SET INDICATES AN ERROR, BIT 10 SET
:6798                                      ;INDICATES CONSOLE CONTROLS ERROR
:6799                                      ;LOOPING, BIT 23 SET INDICATES THAT THE
:6800                                      ;CONSOLE SHOULD NOT PRINT EXPECTED OR
:6801                                      ;RECIEVED DATA ON ERROR REPORT. THIS
:6802                                      ;WORD IS LOADED NOW SO THAT IT NEEDN'T
:6803                                      ;BE LOADED AGAIN IF AN ERROR OCCURS.
U 07E0, 2F80,15 :6804      CLR WR[0]                      ;CLEAR OUT WRO
U 07E1, 2040,15 :6805      INC WR[0]                      ;INITIALIZE
U 07E2, BE82,15 :6806      MOV WR[0] TO LS[ERROR.NUMBER]    ;ERROR NUMBER TO 1
:6807
U 07E3, 2F80,15 :6808      CLR WR[0]                      ;CLEAR WRO SO THAT IT CAN BE USED TO CHECK IF WE
:6809                                      ;ACTUALLY EXECUTED THE SUBROUTINE
U 07E4, 0901,0C :6810      JSR [S.T3.1]                  ;JUMP TO SUBROUTINE
:6811
U 07E5, B6A0,95 :6812      T3.2:
:6813      MOV LS[PREVIOUS.ERROR] TO WR[1] ;GET LAST ERROR NUMBER
:6814      XOR LS[ERROR.NUMBER] WITH WR[1] ;TO Q,
:6815      DT(LONG)&SET.ALU.CC           ;WAS IT THE CURRENT ERROR?
:6816      JMP.IF[NEQ] TO [T3.2A]        ;IF NOT DO NOT CALL ERROR
      MISC [SET.CP.ATTN]              ;REPORT INTERMITTENT ERROR 1
```

```

:6817 WAIT.T3.2A:
U 07E9, 887E,94 :6818 JMP [WAIT.T3.2A] ;WAIT FOR CONSOLE TO RESPOND
U 07EA, 887E,34 :6819 JMP [LOOP.T3.2] ;LOOP ON ERROR IF ENABLED
:6820 T3.2A:
U 07EB, B642,95 :6821 MOV LS[#2] TO WR[1] ;UPDATE ERROR NUMBER
U 07EC, 3E82,95 :6822 MOV WR[1] TO LS[ERROR.NUMBER] ; TO 2
:6823 TST WR[0] ;SEE IF JUMP ACTUALLY OCCURED
U 07ED, 2000,35 :6824 DT(LONG)&SET.ALU.CC
U 07EE, 887F,41 :6825 JMP.IF[NEQ] TO [T3.3] ;GO TO NEXT TEST IF OK
U 07EF, B682,95 :6826 MOV LS[ERROR.NUMBER] TO WR[1] ;ERROR 2 IF WRO DID NOT INCREMENT
U 07F0, 3EAO,95 :6827 MOV WR[1] TO LS[PREVIOUS.ERROR]
U 07F1, 10E0,15 :6828 MISC [SET.CP.ATTN] ;GET CONSOLES ATTENTION
:6829 WAIT.T3.2:
U 07F2, 887F,24 :6830 JMP [WAIT.T3.2] ;WAIT FOR CONSOLE TO RESPOND
U 07F3, 887E,34 :6831 JMP [LOOP.T3.2] ;LOOP ON ERROR IF ENABLED
:6832 T3.3:
U 07F4, 3682,15 :6833 MOV LS[ERROR.NUMBER] TO WR[0] ;GET ERROR NUMBER
U 07F5, 2040,15 :6834 INC WR[0] ; AND
U 07F6, BE82,15 :6835 MOV WR[0] TO LS[ERROR.NUMBER] ; UPDATE TO 3
U 07F7, 2F80,15 :6836 CLR WR[0] ;CLEAR WRO
U 07F8, 09FF,F4 :6837 JMP [LOOP.T3.3]
:6838 1FFF:
:6839 LOOP.T3.3:
U 1FFF, 0901,0C :6840 JSR [S.T3.1] ;JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT13
:6841 TST WR[0]
U 2000, 2000,35 :6842 DT(LONG)&SET.ALU.CC ;SET CC'S ON WRO
U 2001, 0902,A9 :6843 JMP.IF[EQL] TO [ERROR.T3.3] ;IF WE COME HERE WITH WRO CLEARED THIS INDICATES THAT
:6844 ; ONE OF STACK BITS 0-12 FAILED.
U 2002, 3682,15 :6845 MOV LS[ERROR.NUMBER] TO WR[0] ;GET ERROR NUMBER
U 2003, 2040,15 :6846 INC WR[0] ; AND
U 2004, BE82,15 :6847 MOV WR[0] TO LS[ERROR.NUMBER] ; UPDATE TO 4
U 2005, 0AFF,F4 :6848 JMP [LOOP.T3.4]
:6849 2FFF:
:6850 LOOP.T3.4:
U 2FFF, 0901,AC :6851 JSR [S.T3.2] ;JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT12
U 3000, B6AO,95 :6852 MOV LS[PREVIOUS.ERROR] TO WR[1] ;GET PREVIOUS ERROR NUMBER
:6853 XOR LS[ERROR.NUMBER] WITH WR[1] ; TO 0
U 3001, 4D82,B5 :6854 DT(LONG)&SET.ALU.CC ;IS IT THE CURRENT ERROR
U 3002, 8905,D1 :6855 JMP.IF[NEQ] TO [DIS.T3.2] ;GOTO NEXT TEST
U 3003, 10E0,15 :6856 MISC [SET.CP.ATTN] ;REPORT INTERMITTANT ERROR TO CONSOLE
:6857 WAIT.T3.4B:
U 3004, 0B00,44 :6858 JMP [WAIT.T3.4B] ;WAIT FOR CONSOLE TO RESPOND
U 3005, 0903,64 :6859 JMP [DIS.T3.1] ;LOOP ON ERROR IF ENABLED
U 3006, 8905,D4 :6860 JMP [DIS.T3.2] ;GOTO NEXT TEST IF NOT
:6861 T3.5:
U 3007, 3682,15 :6862 MOV LS[ERROR.NUMBER] TO WR[0] ;GET ERROR NUMBER
U 3008, 2040,15 :6863 INC WR[0] ; AND
U 3009, BE82,15 :6864 MOV WR[0] TO LS[ERROR.NUMBER] ; UPDATE TO 5
U 300A, 8A7F,F4 :6865 JMP [LOOP.T3.5]
:6866 27FF:
:6867 LOOP.T3.5:
U 27FF, 0901,AC :6868 JSR [S.T3.2] ;JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT11
U 2800, B6AO,95 :6869 MOV LS[PREVIOUS.ERROR] TO WR[1] ;GET PREVIOUS ERROR NUMBER
:6870 XOR LS[ERROR.NUMBER] WITH WR[1] ; TO 0
U 2801, 4D82,B5 :6871 DT(LONG)&SET.ALU.CC ;IS IT THE CURRENT ERROR

```


ZZ-ENKCA-1.0	:	ENKCA.MIC		F 13			
: ENKCA.MCR			MICRO2 1L(03)	TEST 3 - JSR and St	3-MAR-82	Fiche 1 Frame F13	Sequence 161
: ENKCA.MIC			TEST 3 - JSR and Stack test (including return and return+1)	(M8390 CPU Module)	09:34:33	ENKCA Micro-diagnostic for DAP module	Rev 01.00 Page 155

U 2802, 8905.D1	:6872	JMP IF[NEQ] TO [DIS.T3.2]	:GOTO NEXT TEST
U 2803, 10E0.15	:6873	MISC [SET.CP.ATTN]	:REPORT INTERMITTANT ERROR TO CONSOLE
	:6874	WAIT.T3.5B:	
U 2804, 0A80.44	:6875	JMP [WAIT.T3.5B]	:WAIT FOR CONSOLE TO RESPOND
U 2805, 0903.64	:6876	JMP [DIS.T3.1]	:LOOP ON ERROR IF ENABLED
U 2806, 8905.D4	:6877	JMP [DIS.T3.2]	:GOTO NEXT TEST IF NOT
	:6878	T3.6:	
U 2807, 3682.15	:6879	MOV LS[ERROR.NUMBER] TO WR[0]	:GET ERROR NUMBER
U 2808, 2040.15	:6880	INC WR[0]	: AND
U 2809, BE82.15	:6881	MOV WR[0] TO LS[ERROR.NUMBER]	: UPDATE TO 6
U 280A, 0A3F.F4	:6882	JMP [LOOP.T3.6]	
	:6883	23FF:	
	:6884	LOOP.T3.6:	
U 23FF, 0901.AC	:6885	JSR [S.T3.2]	:JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT10
U 2400, B6A0.95	:6886	MOV LS[PREVIOUS.ERROR] TO WR[1]	:GET PREVIOUS ERROR NUMBER
	:6887	XOR LS[ERROR.NUMBER] WITH WR[1]	TO 0,
U 2401, 4D82.B5	:6888	DT(LONG)&SET.ALU.CC	:IS IT THE CURRENT ERROR
U 2402, 8905.D1	:6889	JMP IF[NEQ] TO [DIS.T3.2]	:GOTO NEXT TEST
U 2403, 10E0.15	:6890	MISC [SET.CP.ATTN]	:REPORT INTERMITTANT ERROR TO CONSOLE
	:6891	WAIT.T3.6B:	
U 2404, 0A40.44	:6892	JMP [WAIT.T3.6B]	:WAIT FOR CONSOLE TO RESPOND
U 2405, 0903.64	:6893	JMP [DIS.T3.1]	:LOOP ON ERROR IF ENABLED
U 2406, 8905.D4	:6894	JMP [DIS.T3.2]	:GOTO NEXT TEST IF NOT
	:6895	T3.7:	
U 2407, 3682.15	:6896	MOV LS[ERROR.NUMBER] TO WR[0]	:GET ERROR NUMBER
U 2408, 2040.15	:6897	INC WR[0]	: AND
U 2409, BE82.15	:6898	MOV WR[0] TO LS[ERROR.NUMBER]	: UPDATE TO 7
U 240A, 8A1F.F4	:6899	JMP [LOOP.T3.7]	
	:6900	21FF:	
	:6901	LOOP.T3.7:	
U 21FF, 0901.AC	:6902	JSR [S.T3.2]	:JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT9
U 2200, B6A0.95	:6903	MOV LS[PREVIOUS.ERROR] TO WR[1]	:GET PREVIOUS ERROR NUMBER
	:6904	XOR LS[ERROR.NUMBER] WITH WR[1]	TO 0,
U 2201, 4D82.B5	:6905	DT(LONG)&SET.ALU.CC	:IS IT THE CURRENT ERROR
U 2202, 8905.D1	:6906	JMP IF[NEQ] TO [DIS.T3.2]	:GOTO NEXT TEST
U 2203, 10E0.15	:6907	MISC [SET.CP.ATTN]	:REPORT INTERMITTANT ERROR TO CONSOLE
	:6908	WAIT.T3.7B:	
U 2204, 0A20.44	:6909	JMP [WAIT.T3.7B]	:WAIT FOR CONSOLE TO RESPOND
U 2205, 0903.64	:6910	JMP [DIS.T3.1]	:LOOP ON ERROR IF ENABLED
U 2206, 8905.D4	:6911	JMP [DIS.T3.2]	:GOTO NEXT TEST IF NOT
	:6912	T3.8:	
U 2207, 3682.15	:6913	MOV LS[ERROR.NUMBER] TO WR[0]	:GET ERROR NUMBER
U 2208, 2040.15	:6914	INC WR[0]	: AND
U 2209, BE82.15	:6915	MOV WR[0] TO LS[ERROR.NUMBER]	: UPDATE TO 8
U 220A, 0A0F.F4	:6916	JMP [LOOP.T3.8]	
	:6917	20FF:	
	:6918	LOOP.T3.8:	
U 20FF, 0901.AC	:6919	JSR [S.T3.2]	:JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT8
U 2100, B6A0.95	:6920	MOV LS[PREVIOUS.ERROR] TO WR[1]	:GET PREVIOUS ERROR NUMBER
	:6921	XOR LS[ERROR.NUMBER] WITH WR[1]	TO 0, ;IS IT THE CURRENT ERROR
U 2101, 4D82.B5	:6922	DT(LONG)&SET.ALU.CC	
U 2102, 8905.D1	:6923	JMP IF[NEQ] TO [DIS.T3.2]	:GOTO NEXT TEST
U 2103, 10E0.15	:6924	MISC [SET.CP.ATTN]	:REPORT INTERMITTANT ERROR TO CONSOLE
	:6925	WAIT.T3.8B:	
U 2104, 0A10.44	:6926	JMP [WAIT.T3.8B]	:WAIT FOR CONSOLE TO RESPOND

G 13
Fiche 1 Frame G13
Sequence 162 Page 156

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

TEST 3 - JSR and St 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 3 - JSR and Stack test (including return and return+1) (M8390 CPU Module)

ENKCA Micro-diagnostic for DAP module
Rev 01.00

```

U 2105, 0903,64 :6927      JMP [DIS.T3.1]          ;LOOP ON ERROR IF ENABLED
U 2106, 8905,D4 :6928      JMP [DIS.T3.2]          ;GOTO NEXT TEST IF NOT
                                T3.9:
U 2107, 3682,15 :6929      MOV LS[ERROR.NUMBER] TO WR[0]      ;GET ERROR NUMBER
U 2108, 2040,15 :6930      INC WR[0]                          ; AND
U 2109, BE82,15 :6931      MOV WR[0] TO LS[ERROR.NUMBER]      ; UPDATE TO 9
U 210A, 8A07,F4 :6932      JMP [LOOP.T3.9]
                                207F:
U 207F, 0901,AC :6933      LOOP.T3.9:
U 2080, B6A0,95 :6934      JSR [S.T3.2]                      ;JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT7
U 2081, 4D82,B5 :6935      MOV LS[PREVIOUS.ERROR] TO WR[1]    ;GET PREVIOUS ERROR NUMBER
U 2082, 8905,D1 :6936      XOR LS[ERROR.NUMBER] WITH WR[1]    ;TO Q, ;IS IT THE CURRENT ERROR
U 2083, 10E0,15 :6937      DT(LONG)&SET.ALU.CC
U 2084, 0A08,44 :6938      JMP.IF[NEQ] TO [DIS.T3.2]          ;GOTO NEXT TEST
U 2085, 0903,64 :6939      MISC [SET.CP.ATTN]                ;REPORT INTERMITTANT ERROR TO CONSOLE
U 2086, 8905,D4 :6940      WAIT.T3.9B:
U 2087, 3682,15 :6941      JMP [WAIT.T3.9B]                  ;WAIT FOR CONSOLE TO RESPOND
U 2088, 2040,15 :6942      JMP [DIS.T3.1]                      ;LOOP ON ERROR IF ENABLED
U 2089, BE82,15 :6943      JMP [DIS.T3.2]                      ;GOTO NEXT TEST IF NOT
U 208A, 0A03,F4 :6944      T3.A:
U 208B, 3682,15 :6945      MOV LS[ERROR.NUMBER] TO WR[0]      ;GET ERROR NUMBER
U 208C, 2040,15 :6946      INC WR[0]                          ; AND
U 208D, BE82,15 :6947      MOV WR[0] TO LS[ERROR.NUMBER]      ; UPDATE TO 10
U 208E, 0A03,F4 :6948      JMP [LOOP.T3.A]
                                203F:
U 203F, 0901,AC :6949      LOOP.T3.A:
U 2040, B6A0,95 :6950      JSR [S.T3.2]                      ;JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT6
U 2041, 4D82,B5 :6951      MOV LS[PREVIOUS.ERROR] TO WR[1]    ;GET PREVIOUS ERROR NUMBER
U 2042, 8905,D1 :6952      XOR LS[ERROR.NUMBER] WITH WR[1]    ;TO Q, ;IS IT THE CURRENT ERROR
U 2043, 10E0,15 :6953      DT(LONG)&SET.ALU.CC
U 2044, 0A04,44 :6954      JMP.IF[NEQ] TO [DIS.T3.2]          ;GOTO NEXT TEST
U 2045, 0903,64 :6955      MISC [SET.CP.ATTN]                ;REPORT INTERMITTANT ERROR TO CONSOLE
U 2046, 8905,D4 :6956      WAIT.T3.AB:
U 2047, 3682,15 :6957      JMP [WAIT.T3.AB]                  ;WAIT FOR CONSOLE TO RESPOND
U 2048, 2040,15 :6958      JMP [DIS.T3.1]                      ;LOOP ON ERROR IF ENABLED
U 2049, BE82,15 :6959      JMP [DIS.T3.2]                      ;GOTO NEXT TEST IF NOT
U 204A, 8A01,F4 :6960      T3.B:
U 204B, 3682,15 :6961      MOV LS[ERROR.NUMBER] TO WR[0]      ;GET ERROR NUMBER
U 204C, 2040,15 :6962      INC WR[0]                          ; AND
U 204D, BE82,15 :6963      MOV WR[0] TO LS[ERROR.NUMBER]      ; UPDATE TO 11
U 204E, 8A01,F4 :6964      JMP [LOOP.T3.B]
                                201F:
U 201F, 0901,AC :6965      LOOP.T3.B:
U 2020, B6A0,95 :6966      JSR [S.T3.2]                      ;JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT5
U 2021, 4D82,B5 :6967      MOV LS[PREVIOUS.ERROR] TO WR[1]    ;GET PREVIOUS ERROR NUMBER
U 2022, 8905,D1 :6968      XOR LS[ERROR.NUMBER] WITH WR[1]    ;TO Q, ;IS IT THE CURRENT ERROR
U 2023, 10E0,15 :6969      DT(LONG)&SET.ALU.CC
U 2024, 0A02,44 :6970      JMP.IF[NEQ] TO [DIS.T3.2]          ;GOTO NEXT TEST
U 2025, 0903,64 :6971      MISC [SET.CP.ATTN]                ;REPORT INTERMITTANT ERROR TO CONSOLE
U 2026, 8905,D4 :6972      WAIT.T3.BB:
U 2027, 3682,15 :6973      JMP [WAIT.T3.BB]                  ;WAIT FOR CONSOLE TO RESPOND
                                T3.C:
U 2028, 2040,15 :6974      JMP [DIS.T3.1]                      ;LOOP ON ERROR IF ENABLED
U 2029, BE82,15 :6975      JMP [DIS.T3.2]                      ;GOTO NEXT TEST IF NOT
U 202A, 3682,15 :6976      MOV LS[ERROR.NUMBER] TO WR[0]      ;GET ERROR NUMBER

```

```

U 2028, 2040,15 :6982      INC WR[0]                : AND
U 2029, BE82,15 :6983      MOV WR[0] TO LS[ERROR.NUMBER] : UPDATE TO 12
U 202A, 0A00,F4 :6984      JMP [LOOP.T3.C]
:6985
200F:
LOOP.T3.C:
U 200F, 0901,AC :6987      JSR [S.T3.2]                :JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT4
U 2010, B6A0,95 :6988      MOV LS[PREVIOUS.ERROR] TO WR[1] :GET PREVIOUS ERROR NUMBER
:6989      XOR LS[ERROR.NUMBER] WITH WR[1] TO Q, :IS IT THE CURRENT ERROR
U 2011, 4D82,B5 :6990      DT(LONG)&SET.ALU.CC
U 2012, 8905,D1 :6991      JMP.IF[NEQ] TO [DIS.T3.2]        :GOTO NEXT TEST
U 2013, 10E0,15 :6992      MISC [SET.CP.ATTN]                :REPORT INTERMITTANT ERROR TO CONSOLE
:6993
WAIT.T3.CB:
U 2014, 0A01,44 :6994      JMP [WAIT.T3.CB]                :WAIT FOR CONSOLE TO RESPOND
U 2015, 0903,64 :6995      JMP [DIS.T3.1]                :LOOP ON ERROR IF ENABLED
U 2016, 8905,D4 :6996      JMP [DIS.T3.2]                :GOTO NEXT TEST IF NOT
:6997
T3.D:
U 2017, 3682,15 :6998      MOV LS[ERROR.NUMBER] TO WR[0]        :GET ERROR NUMBER
U 2018, 2040,15 :6999      INC WR[0]                : AND
U 2019, BE82,15 :7000      MOV WR[0] TO LS[ERROR.NUMBER]    : UPDATE TO 13
U 201A, 8A30,74 :7001      JMP [LOOP.T3.D]
:7002
2307:
LOOP.T3.D:
U 2307, 0901,AC :7004      JSR [S.T3.2]                :JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT3
U 2308, B6A0,95 :7005      MOV LS[PREVIOUS.ERROR] TO WR[1] :GET PREVIOUS ERROR NUMBER
:7006      XOR LS[ERROR.NUMBER] WITH WR[1] TO Q, :IS IT THE CURRENT ERROR
U 2309, 4D82,B5 :7007      DT(LONG)&SET.ALU.CC
U 230A, 8905,D1 :7008      JMP.IF[NEQ] TO [DIS.T3.2]        :GOTO NEXT TEST
U 230B, 10E0,15 :7009      MISC [SET.CP.ATTN]                :REPORT INTERMITTANT ERROR TO CONSOLE
:7010
WAIT.T3.DB:
U 230C, 0A30,C4 :7011      JMP [WAIT.T3.DB]                :WAIT FOR CONSOLE TO RESPOND
U 230D, 0903,64 :7012      JMP [DIS.T3.1]                :LOOP ON ERROR IF ENABLED
U 230E, 8905,D4 :7013      JMP [DIS.T3.2]                :GOTO NEXT TEST IF NOT
:7014
T3.E:
U 230F, 3682,15 :7015      MOV LS[ERROR.NUMBER] TO WR[0]        :GET ERROR NUMBER
U 2310, 2040,15 :7016      INC WR[0]                : AND
U 2311, BE82,15 :7017      MOV WR[0] TO LS[ERROR.NUMBER]    : UPDATE TO 14
U 2312, 0A50,34 :7018      JMP [LOOP.T3.E]
:7019
2503:
LOOP.T3.E:
U 2503, 0901,AC :7021      JSR [S.T3.2]                :JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT2
U 2504, B6A0,95 :7022      MOV LS[PREVIOUS.ERROR] TO WR[1] :GET PREVIOUS ERROR NUMBER
:7023      XOR LS[ERROR.NUMBER] WITH WR[1] TO Q, :IS IT THE CURRENT ERROR
U 2505, 4D82,B5 :7024      DT(LONG)&SET.ALU.CC
U 2506, 8905,D1 :7025      JMP.IF[NEQ] TO [DIS.T3.2]        :GOTO NEXT TEST
U 2507, 10E0,15 :7026      MISC [SET.CP.ATTN]                :REPORT INTERMITTANT ERROR TO CONSOLE
:7027
WAIT.T3.EB:
U 2508, 8A50,84 :7028      JMP [WAIT.T3.EB]                :WAIT FOR CONSOLE TO RESPOND
U 2509, 0903,64 :7029      JMP [DIS.T3.1]                :LOOP ON ERROR IF ENABLED
U 250A, 8905,D4 :7030      JMP [DIS.T3.2]                :GOTO NEXT TEST IF NOT
:7031
T3.F:
U 250B, 3682,15 :7032      MOV LS[ERROR.NUMBER] TO WR[0]        :GET ERROR NUMBER
U 250C, 2040,15 :7033      INC WR[0]                : AND
U 250D, BE82,15 :7034      MOV WR[0] TO LS[ERROR.NUMBER]    : UPDATE TO 15
U 250E, 8A60,14 :7035      JMP [LOOP.T3.F]
:7036
2601:
    
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) TEST 3 - JSR and St 3-MAR-82 09:34:33
TEST 3 - JSR and Stack test (including return and return+1) (M8390 CPU Module)

Fiche 1 Frame I13

Sequence 164
Rev 01.00 Page 158

```
U 2601, 0901,AC :7037
U 2602, B6A0,95 :7038
U 2603, 4D82,B5 :7039
U 2604, 8905,D1 :7040
U 2605, 10E0,15 :7041
U 2606, 0A60,64 :7042
U 2607, 0903,64 :7043
U 2608, 8905,D4 :7044
U 2609, 3682,15 :7045
U 260A, 2040,15 :7046
U 260B, BE82,15 :7047
U 260C, 8A70,04 :7048
U 260D, 2F82,95 :7049
U 2700, 0902,5C :7050
U 2701, B6A0,95 :7051
U 2702, 4D82,B5 :7052
U 2703, 8905,D1 :7053
U 2704, 10E0,15 :7054
U 2705, 8A70,54 :7055
U 2706, 0903,64 :7056
U 2707, 8905,D4 :7057
U 2708, 3682,15 :7058
U 2709, 2040,15 :7059
U 270A, BE82,15 :7060
U 270B, 2F80,15 :7061
U 270C, 8901,EC :7062
U 270D, 2040,15 :7063
U 270E, B6A0,95 :7064
U 270F, 4D82,B5 :7065
U 2710, 0A71,51 :7066
U 2711, 10E0,15 :7067
U 2712, 8A71,24 :7068
U 2713, 0A70,B4 :7069
U 2714, 0981,04 :7070
U 2715, 36C2,95 :7071
U 2716, 3E82,95 :7072
U 2717, 2000,35 :7073
U 2718, 0903,01 :7074
U 2719, 0981,04 :7075
U 2300, 8902,A4 :7076
:7077
:7078
:7079
:7080
:7081
:7082
:7083
:7084
:7085
:7086
:7087
:7088
:7089
:7090
:7091

LOOP.T3.F:
JSR [S,T3.2] :JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT1
MOV LS[PREVIOUS.ERROR] TO WR[1] :GET PREVIOUS ERROR NUMBER
XOR LS[ERROR.NUMBER] WITH WR[1] :TO Q, ;IS IT THE CURRENT ERROR
DT(LONG)&SET.ALU.CC
JMP.IF[NEQ] TO [DIS.T3.2] :GOTO NEXT TEST
MISC [SET.CP.ATTN] :REPORT INTERMITTANT ERROR TO CONSOLE

WAIT.T3.FB:
JMP [WAIT.T3.FB] :WAIT FOR CONSOLE TO RESPOND
JMP [DIS.T3.1] :LOOP ON ERROR IF ENABLED
JMP [DIS.T3.2] :GOTO NEXT TEST IF NOT

T3.10:
MOV LS[ERROR.NUMBER] TO WR[0] :GET ERROR NUMBER
INC WR[0] :AND
MOV WR[0] TO LS[ERROR.NUMBER] :UPDATE TO 16
JMP [LOOP.T3.10]
CLR WR[1] :CLEAR COUNTER

2700:
LOOP.T3.10:
JSR [S,T3.10] :JUMP TO SUBROUTINE FROM ADDRESS THAT WILL CHECK BIT0
MOV LS[PREVIOUS.ERROR] TO WR[1] :GET PREVIOUS ERROR NUMBER
XOR LS[ERROR.NUMBER] WITH WR[1] :TO Q, ;IS IT THE CURRENT ERROR
DT(LONG)&SET.ALU.CC
JMP.IF[NEQ] TO [DIS.T3.2] :GOTO NEXT TEST
MISC [SET.CP.ATTN] :REPORT INTERMITTANT ERROR TO CONSOLE

WAIT.T3.10B:
JMP [WAIT.T3.10B] :WAIT FOR CONSOLE TO RESPOND
JMP [DIS.T3.1] :LOOP ON ERROR IF ENABLED
JMP [DIS.T3.2] :GOTO NEXT TEST IF NOT

T3.11:
MOV LS[ERROR.NUMBER] TO WR[0] :GET ERROR NUMBER
INC WR[0] :AND
MOV WR[0] TO LS[ERROR.NUMBER] :UPDATE TO 17

LOOP.T3.12:
CLR WR[0] :CLEAR WRO
JSR [S,T3.3] :JUMP TO RTN+1 ROUTINE
INC WR[0] :ERROR IF EXECUTED
MOV LS[PREVIOUS.ERROR] TO WR[1] :GET PREVIOUS ERROR NUMBER
XOR LS[ERROR.NUMBER] WITH WR[1] :TO Q, ;IS IT THE CURRENT ERROR
DT(LONG)&SET.ALU.CC
JMP.IF[NEQ] TO [T3.12] :GOTO END OF TEST IF NO ERROR 17
MISC [SET.CP.ATTN] :REPORT INTERMITTANT ERROR 17 TO CONSOLE

WAIT.T3.11A:
JMP [WAIT.T3.11A] :WAIT FOR CONSOLE TO RESPOND
JMP [LOOP.T3.12] :LOOP ON ERROR IF ENABLED
JMP [END.T3] :GOTO END OF TEST

T3.12:
MOV LS[#12(H)] TO WR[1] :UPDATE ERROR NUMBER
MOV WR[1] TO LS[ERROR.NUMBER] :TO 18
TST WR[0]
DT(LONG)&SET.ALU.CC :SEE IF INSTRUCTION AFTER JSR WAS EXECUTED
JMP.IF[NEQ] TO [ERROR.T3.12] :GO TO ERROR ROUTINE IF SO
JMP [END.T3] :SKIP OVER SUB AND ERROR ROUTINES

2300:
JMP [ERROR.T3.3] :BIT3 IN ERROR
```


U 2500, 8902,A4	:7092	2500:	JMP [ERROR.T3.3]	:BIT2 IN ERROR
U 2600, 8902,A4	:7093	2600:	JMP [ERROR.T3.3]	:BIT1 IN ERROR
U 1010, 10D0,15	:7094	1010:		
U 1011, 90C0,15	:7095	S.T3.1:	MISC [SET.CP.ACK]	:SCOPE SYNC POINT
U 1012, 2040,15	:7096		MISC [CLR.CP.ATTN.AND.ACK]	:CLEAR SIGNALS FOR SYNCING
U 1013, 5B00,14	:7097		INC WR[0]	:SET WRO TO A 1 SO THAT WE CAN TELL WE'VE BEEN HERE
U 1014, B682,95	:7098		RETURN	:RETURN
U 1015, 3EA0,95	:7099		MOV LS[ERROR.NUMBER] TO WR[1]	:ERROR IF FALLEN THROUGH TO HERE
U 1016, 10E0,15	:7100		MOV WR[1] TO LS[PREVIOUS.ERROR]	:SET PREVIOUS ERROR
U 1017, 0901,74	:7101		MISC [SET.CP.ATTN]	:REPORT ERROR 1 TO CONSOLE
U 1018, 887E,34	:7102	WAIT.T3.1:	JMP [WAIT.T3.1]	:WAIT FOR CONSOLE TO RESPOND
U 1019, 087E,B4	:7103		JMP [LOOP.T3.2]	:LOOP ON ERROR IF ENABLED
U 101A, 10D0,15	:7104		JMP [T3.2A]	:GOTO NEXT TEST
U 101B, 90C0,15	:7105	S.T3.2:	MISC [SET.CP.ACK]	:SCOPE SYNC POINT
U 101C, 2F80,15	:7106		MISC [CLR.CP.ATTN.AND.ACK]	:CLEAR SIGNALS FOR SYNCING
U 101D, 5B00,14	:7107		CLR WR[0]	:INSURE WRO IS CLEARED SO THAT IF THE BIT BEING CHECKED
U 101E, DB00,16	:7108		RETURN	:FAILS WE WILL KNOW IT IS AN ERROR
U 101F, B682,95	:7109	S.T3.3:	RETURN+1	:RETURN
U 1020, 3EA0,95	:7110		MOV LS[ERROR.NUMBER] TO WR[1]	:RETURN AND SKIP
U 1021, 10E0,15	:7111		MOV WR[1] TO LS[PREVIOUS.ERROR]	:
U 1022, 0902,24	:7112		MISC [SET.CP.ATTN]	:ERROR IF FALLEN THROUGH TO HERE
U 1023, 0901,E4	:7113	WAIT.T3.11:	JMP [WAIT.T3.11]	:WAIT FOR CONSOLE TO RESPOND
U 1024, 0981,04	:7114		JMP [S.T3.3]	:LOOP ON ERROR IF ENABLED
U 1025, 2F80,15	:7115		JMP [END.T3]	:GO TO END OF TEST SINCE THE RETURN+1 DOES NOT WORK
U 1026, 2042,95	:7116	S.T3.10:	CLR WR[0]	:INSURE THAT WRO IS CLEARED
U 1027, 4D42,B5	:7117		INC WR[1]	:INCREMENT COUNTER
U 1028, 0902,A9	:7118		XOR LS[BIT1] WITH WR[1] TO Q,	:IF COUNTER GOES ABOVE 1 BIT0 IN ERROR
U 1029, 5B00,14	:7119		DT(LONG)&SET.ALU.CC	
U 102A, B682,95	:7120		JMP.[IF[EQL] TO [ERROR.T3.3]	:GO TO ERROR ROUTINE IF BIT0 IN ERROR
U 102B, 3EA0,95	:7121	ERROR.T3.3:	RETURN	
U 102C, 10E0,15	:7122		MOV LS[ERROR.NUMBER] TO WR[1]	:MOVE ERROR NUMBER
U 102D, 0902,D4	:7123		MOV WR[1] TO LS[PREVIOUS.ERROR]	:TO PREVIOUS ERROR
U 102E, 0903,64	:7124		MISC [SET.CP.ATTN]	:TELL CONSOLE A BIT HAS FAILED
U 102F, 8905,D4	:7125	WAIT.T3.3:	JMP [WAIT.T3.3]	:WAIT FOR CONSOLE TO RESPOND
U 1030, B682,95	:7126		JMP [DIS.T3.1]	:LOOP ON ERROR IF ENABLED
U 1031, 3EA0,95	:7127		JMP [DIS.T3.2]	:GOTO NEXT TEST
U 1032, 10E0,15	:7128	ERROR.T3.12:	MOV LS[ERROR.NUMBER] TO WR[1]	:
U 1033, 0903,34	:7129		MOV WR[1] TO LS[PREVIOUS.ERROR]	:
U 1034, 0A70,B4	:7130		MISC [SET.CP.ATTN]	:TELL CONSOLE RETURN+1 FAILED TO SKIP
U 1035, 0908,44	:7131	WAIT.T3.12:	JMP [WAIT.T3.12]	:WAIT FOR CONSOLE TO RESPOND
	:7132		JMP [LOOP.T3.12]	:LOOP ON ERROR IF ENABLED
	:7133		JMP [T3.12A]	:GOTO END OF TEST

```

DIS.T3.1:
U 1036, 3644,15 :7147 MOV LS[#4] TO WR[0] ;SET WRO TO 4
:7148 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR 4
:7149 DT(LONG)&SET.ALU.CC
U 1037, CD82,35 :7150 JMP.IF[EQL] TO [LOOP.T3.4] ;LOOP ON ERROR IF SO
U 1038, 8AFF,F9 :7151 INC WR[0] ;TRY NEXT ERROR IF NOT
U 1039, 2040,15 :7152 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR 5
:7153 DT(LONG)&SET.ALU.CC
U 103A, CD82,35 :7154 JMP.IF[EQL] TO [LOOP.T3.5] ;LOOP ON ERROR IF SO
U 103B, 0A7F,F9 :7155 INC WR[0] ;TRY NEXT ERROR IF NOT
U 103C, 2040,15 :7156 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR 6
:7157 DT(LONG)&SET.ALU.CC
U 103D, CD82,35 :7158 JMP.IF[EQL] TO [LOOP.T3.6] ;LOOP ON ERROR IF SO
U 103E, 8A3F,F9 :7159 INC WR[0] ;TRY NEXT ERROR IF NOT
U 103F, 2040,15 :7160 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR 7
:7161 DT(LONG)&SET.ALU.CC
U 1040, CD82,35 :7162 JMP.IF[EQL] TO [LOOP.T3.7] ;LOOP ON ERROR IF SO
U 1041, 0A1F,F9 :7163 INC WR[0] ;TRY NEXT ERROR IF NOT
U 1042, 2040,15 :7164 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR 8
:7165 DT(LONG)&SET.ALU.CC
U 1043, CD82,35 :7166 JMP.IF[EQL] TO [LOOP.T3.8] ;LOOP ON ERROR IF SO
U 1044, 8A0F,F9 :7167 INC WR[0] ;TRY NEXT ERROR IF NOT
U 1045, 2040,15 :7168 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR 9
:7169 DT(LONG)&SET.ALU.CC
U 1046, CD82,35 :7170 JMP.IF[EQL] TO [LOOP.T3.9] ;LOOP ON ERROR IF SO
U 1047, 0A07,F9 :7171 INC WR[0] ;TRY NEXT ERROR IF NOT
U 1048, 2040,15 :7172 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR A
:7173 DT(LONG)&SET.ALU.CC
U 1049, CD82,35 :7174 JMP.IF[EQL] TO [LOOP.T3.A] ;LOOP ON ERROR IF SO
U 104A, 8A03,F9 :7175 INC WR[0] ;TRY NEXT ERROR IF NOT
U 104B, 2040,15 :7176 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR B
:7177 DT(LONG)&SET.ALU.CC
U 104C, CD82,35 :7178 JMP.IF[EQL] TO [LOOP.T3.B] ;LOOP ON ERROR IF SO
U 104D, 0A01,F9 :7179 INC WR[0] ;TRY NEXT ERROR IF NOT
U 104E, 2040,15 :7180 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR C
:7181 DT(LONG)&SET.ALU.CC
U 104F, CD82,35 :7182 JMP.IF[EQL] TO [LOOP.T3.C] ;LOOP ON ERROR IF SO
U 1050, 8A00,F9 :7183 INC WR[0] ;TRY NEXT ERROR IF NOT
U 1051, 2040,15 :7184 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR D
:7185 DT(LONG)&SET.ALU.CC
U 1052, CD82,35 :7186 JMP.IF[EQL] TO [LOOP.T3.D] ;LOOP ON ERROR IF SO
U 1053, 0A30,79 :7187 INC WR[0] ;TRY NEXT ERROR IF NOT
U 1054, 2040,15 :7188 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR E
:7189 DT(LONG)&SET.ALU.CC
U 1055, CD82,35 :7190 JMP.IF[EQL] TO [LOOP.T3.E] ;LOOP ON ERROR IF SO
U 1056, 8A50,39 :7191 INC WR[0] ;TRY NEXT ERROR IF NOT
U 1057, 2040,15 :7192 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR F
:7193 DT(LONG)&SET.ALU.CC
U 1058, CD82,35 :7194 JMP.IF[EQL] TO [LOOP.T3.F] ;LOOP ON ERROR IF SO
U 1059, 0A60,19 :7195 INC WR[0] ;TRY NEXT ERROR IF NOT
U 105A, 2040,15 :7196 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;IS THIS ERROR 10
:7197 DT(LONG)&SET.ALU.CC
U 105B, CD82,35 :7198 JMP.IF[EQL] TO [LOOP.T3.10] ;LOOP ON ERROR IF SO
U 105C, 0A70,09 :7199
:7200
U 105D, 3644,15 :7201 MOV LS[#4] TO WR[0] ;SET WRO TO 4
  
```


ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

L 13
TEST 3 - JSR and St. 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 3 - JSR and Stack test (including return and return+1) (M8390 CPU Module)

Fiche 1 Frame L13
Sequence 167
Rev 01.00 Page 161

U 105E, CD82,35	:7202	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR 4
U 105F, 8B00,79	:7203	DT(LONG)&SET.ALU.CC	
U 1060, 2040,15	:7204	JMP.IF[EQL] TO [T3.5]	:IF SO GOTO TEST 5
	:7205	INC WR[0]	:SET WRO TO 5
U 1061, CD82,35	:7206	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR 5
U 1062, 8A80,79	:7207	DT(LONG)&SET.ALU.CC	
U 1063, 2040,15	:7208	JMP.IF[EQL] TO [T3.6]	:IF SO GOTO TEST 6
	:7209	INC WR[0]	:SET WRO TO 6
U 1064, CD82,35	:7210	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR 6
U 1065, 8A40,79	:7211	DT(LONG)&SET.ALU.CC	
U 1066, 2040,15	:7212	JMP.IF[EQL] TO [T3.7]	:IF SO GOTO TEST 7
	:7213	INC WR[0]	:SET WRO TO 7
U 1067, CD82,35	:7214	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR 7
U 1068, 8A20,79	:7215	DT(LONG)&SET.ALU.CC	
U 1069, 2040,15	:7216	JMP.IF[EQL] TO [T3.8]	:IF SO GOTO TEST 8
	:7217	INC WR[0]	:SET WRO TO 8
U 106A, CD82,35	:7218	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR 8
U 106B, 8A10,79	:7219	DT(LONG)&SET.ALU.CC	
U 106C, 2040,15	:7220	JMP.IF[EQL] TO [T3.9]	:IF SO GOTO TEST 9
	:7221	INC WR[0]	:SET WRO TO 9
U 106D, CD82,35	:7222	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR 9
U 106E, 8A08,79	:7223	DT(LONG)&SET.ALU.CC	
U 106F, 2040,15	:7224	JMP.IF[EQL] TO [T3.A]	:IF SO GOTO TEST A
	:7225	INC WR[0]	:SET WRO TO A
U 1070, CD82,35	:7226	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR A
U 1071, 8A04,79	:7227	DT(LONG)&SET.ALU.CC	
U 1072, 2040,15	:7228	JMP.IF[EQL] TO [T3.B]	:IF SO GOTO TEST B
	:7229	INC WR[0]	:SET WRO TO B
U 1073, CD82,35	:7230	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR B
U 1074, 8A02,79	:7231	DT(LONG)&SET.ALU.CC	
U 1075, 2040,15	:7232	JMP.IF[EQL] TO [T3.C]	:IF SO GOTO TEST C
	:7233	INC WR[0]	:SET WRO TO C
U 1076, CD82,35	:7234	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR C
U 1077, 8A01,79	:7235	DT(LONG)&SET.ALU.CC	
U 1078, 2040,15	:7236	JMP.IF[EQL] TO [T3.D]	:IF SO GOTO TEST D
	:7237	INC WR[0]	:SET WRO TO D
U 1079, CD82,35	:7238	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR D
U 107A, 8A30,F9	:7239	DT(LONG)&SET.ALU.CC	
U 107B, 2040,15	:7240	JMP.IF[EQL] TO [T3.E]	:IF SO GOTO TEST E
	:7241	INC WR[0]	:SET WRO TO E
U 107C, CD82,35	:7242	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR E
U 107D, 0A50,B9	:7243	DT(LONG)&SET.ALU.CC	
U 107E, 2040,15	:7244	JMP.IF[EQL] TO [T3.F]	:IF SO GOTO TEST F
	:7245	INC WR[0]	:SET WRO TO F
U 107F, CD82,35	:7246	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR F
U 1080, 8A60,99	:7247	DT(LONG)&SET.ALU.CC	
U 1081, 2040,15	:7248	JMP.IF[EQL] TO [T3.10]	:IF SO GOTO TEST 10
	:7249	INC WR[0]	:SET WRO TO 10
U 1082, CD82,35	:7250	XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,	:IS THIS ERROR 10
U 1083, 8A70,89	:7251	DT(LONG)&SET.ALU.CC	
	:7252	JMP.IF[EQL] TO [T3.11]	:IF SO GOTO TEST 11
U 1084, B6A0,95	:7253		
U 1085, C382,B5	:7254		
	:7255		
	:7256		

T3.12A:

MOV LS[PREVIOUS.ERROR] TO WR[1] ;GET PREVIOUS ERROR NUMBER
XOR LS[ERROR.NUMBER] TO WR[1], ;IS IT THE CURRENT ERROR
DT(LONG)&SET.ALU.CC

ZZ-ENKCA-1.0	:	ENKCA.MIC							
:	:	ENKCA.MCR	MICRO2	1L(03)	TEST 3 - JSR and St	M 13	Fiche 1	Frame M13	Sequence 168
:	:	ENKCA.MIC			3-MAR-82	09:34:33	ENKCA Micro-diagnostic for DAP module	Rev 01.00	Page 162
					TEST 3 - JSR and Stack test (including return and return+1) (M8390 CPU Module)				

U 1086, 0981,01	:	7257	JMP IF[NEQ] TO [END.T3]	:GOTO END OF TEST IF NO ERROR 18
U 1087, 10E0,15	:	7258	MISC [SET.CP.ATTN]	:REPORT INTERMITTANT ERROR 18 TO CONSOLE
	:	7259	WAIT.T3.12A:	
U 1088, 0908,84	:	7260	JMP [WAIT.T3.12A]	:WAIT FOR CONSOLE TO RESPOND
U 1089, 0A70,B4	:	7261	JMP [LOOP.T3.12]	:LOOP ON ERROR IF ENABLED
	:	7262		
	:	7263	1810:	
	:		END.T3:	

Page "TEST 4 - Index Mode Into LS At Speed test (M8390 CPU Module)"

++

Test Description:

This test checks addresses A0(H)-BF(H) in the LS using index mode. The LS has been completely checked via the console previously, but this is the first LS test executed at speed. The test will check locations A0(H) through BF(H) with a shifting 1 pattern and also run a march test to check the addressing logic. The data path is as follows: From the 2901 to the LS via the Y bus, through the 8 bit latch to the D bus, and back to the 2901 via the D bus for checking. The LS address is generated by first loading the OS reg via the Y bus. The OS reg bits 0-4 are selected at the 2 to 1 mux forming LS ADRS 0 L through LS ADRS 4 L. LS ADRS 5, 6, and 7 are formed by CSR bits in the MOV micro-instruction. The OS will be written in the micro-state just preceeding the write or read of LS to assure that any excessive propagation delays are detected.

Assumptions:

It is assumed that the LS has been previously tested for integrity by the console and that XOR WR WITH LS TO Q has also been tested and is working correctly. Since the LS is used by the console error routine, locations 40-46 must be written with pertinent data before calling the console to report an error. So the data that is in error may be overwritten before the console prints the error. The printout will reflect the actual data conditions that caused the error however.

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Set up the OS reg and execute a write to LS using index mode (address in range 20-3F) with data of all zeros and check the result.
- 3) Set up the OS reg and execute a write to LS using index mode (address in range 40-4F) with data of all zeros and check the result.
- 4) INC the 2901 to produce a shifting one pattern and write the same location in LS and check this result.
- 5) Repeat step 2 until all 32 bits have been tested.
- 6) Increment OS to point to the next LS address and repeat steps 2 and 3 until locations A0-BF have been checked.
- 7) INC the 2901 to produce a shifting one pattern and write the same location in LS and check this result.
- 8) Repeat step 2 until all 32 bits have been tested.
- 9) Increment OS to point to the next LS address and repeat steps 2 and 3 until locations 80-8F have been checked.
- 10) Write a background of zeros in LS A0-BF.
- 11) Read a zero at the first location of LS (address A0), write all ones, and read all ones. Inc the address and repeat this step until locations A0-BF have been checked.
- 12) Starting at location A0, read ones, write zeros, and read zeros. Then increment the address and repeat this step until locations A0-BF have been checked.

Errors:


```

:7319
:7320 Error 1 - LS Adress failure Location in range 20-3F (hex).
:7321 Error 2 - LS Adress failure Location in range 40-4F (hex).
:7322 Error 3 - LS failure with shifting data. Location A0-BF (hex).
:7323 Error 4 - LS failure with shifting data. Location 80-8F (hex).
:7324 Error 5 - LS address failure (march test). Ascending address, Location
:7325 A0-BF (hex).
:7326 Error 6 - LS address failure (march test). Ascending address, Location
:7327 A0-BF (hex).
:7328
:7329
:7330

```

Debug:

```

:7331 Error 1 - This failure indicates that the data was written to the wrong
:7332 location using Index Mode. Check the logic associated with indexing.
:7333 Failing Address is OS bits 4-0 plus 20(H).
:7334
:7335 Error 2 - This failure indicates that the data was written to the wrong
:7336 location using Index Mode. Check the logic associated with indexing.
:7337 Failing Address is OS bits 3-0 plus 40(H).
:7338
:7339 Error 3 - This failure indicates a possible timing problem or slow
:7340 chip. This path had been previously tested by the console. The only
:7341 difference between this test and the one run by the console is the
:7342 execution speed. One other possibility is that COMPAT MODE failed to
:7343 get cleared during initialization and is forcing a word write to LS.
:7344 If only the upper 16 bits are in error, check that COMPAT MODE H going
:7345 to the CC CONTROL PAL is low (this can be checked statically using single
:7346 step). Otherwise, follow the path described above while looping on
:7347 error or test. To determine the failing LS address, examine OS bits
:7348 0-4 and add 20(H).
:7349
:7350 Error 4 - This failure may be caused by CSR 09 H failing to get through
:7351 the 2 to 1 mux feeding LS ADRS 5 and 6. This path was not tested in
:7352 the low state by the console test. CSR 09 should be low for LS addr-
:7353 esses 40-4F(H) and cause LS ADRS 5 L to be high and LS ADRS 6 L to be
:7354 low. If these are OK, the problem is probably a timing one like in
:7355 error 3. The LS address that failed can be determined by examining OS
:7356 bits 0-3 and adding 40(H).
:7357
:7358 Error 5 - A failure here indicates a possible timing or slow chip as in
:7359 error 3. OS bits 0-4 plus 20(H) is the failing address.
:7360
:7361 Error 6 - Same as error 5
:7362
:7363
:7364
:7365
:7366

```

```

U 1810, B65E,15 :7367
U 1811, 3E80,15 :7368
U 1812, 10E0,15 :7369
U 1813, 0981,34 :7372
U 1814, 2F80,15 :7373

T.4:
  MOV LS[BIT15] TO WR[0] ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
  MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CONTROL & STATUS WORD
  MISC [SET.CP.ATTN] ;BIT15 INDICATES START OF TEST
  ;CALL CONSOLE
WAIT.T4.0:
  JMP [WAIT.T4.0] ;WAIT FOR CONSOLE TO RESPOND
  CLR WR[0]

```


ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

C 14
TEST 4 - Index Mode 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module Rev 01.00
TEST 4 - Index Mode Into LS At Speed test (M8390 CPU Module) Fiche 1 Frame C14 Sequence 171 Page 165

```
U 1815, 3E8A,15 :7374      MOV WR[0] TO LS[ERROR.MASK]      ;CLEAR ERROR MASK
U 1816, B63E,15 :7375      MOV LS[ERR.CON] TO WR[0]      ;WRO = BIT8:BIT10
U 1817, 3E80,15 :7376      MOV WR[0] TO LS[CONTROL.STATUS] ;LOAD CONTROL& STATUS WORD
                        :7377      ; BIT8 = ERROR
                        :7378      ; BIT10 = CONSOLE CONTROLS LOOP ON ERROR
U 1818, 2F80,15 :7379      CLR WR[0]
U 1819, BEA0,15 :7380      MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR
U 181A, 3EF8,15 :7381      MOV WR[0] TO LS[OS]      ;CLEAR THE OS REGISTER
U 181B, 367E,15 :7382      MOV LS[BIT31] TO WR[0]
U 181C, 3E10,15 :7383      MOV WR[0] TO LS[T8]      ;T8 = BIT31
U 181D, B64A,15 :7384      MOV LS[#20] TO WR[0]
U 181E, BE12,15 :7385      MOV WR[0] TO LS[T9]      ;T9 = 32(DECIMAL)
U 181F, 3660,15 :7386      MOV LS[BIT16] TO WR[0]
U 1820, BE14,15 :7387      MOV WR[0] TO LS[T10]     ;T10 = BIT 16
U 1821, B640,15 :7388      MOV LS[#1] TO WR[0]
U 1822, BE18,15 :7389      MOV WR[0] TO LS[T12]
U 1823, BE82,15 :7390      MOV WR[0] TO LS[ERROR.NUMBER] ;SAVE A ONE
                        :7391      ;ERROR NUMBER = 1
U 1824, 3698,95 :7391      MOV LS[ALTER.ODD] TO WR[1] ;WR1 = PATTERN
                        :7392
T4.1:      JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC
U 1825, 8870,9C :7393      MOV WR[1] TO LS[SHIFT.OS(4-0)] ;WRITE PATTERN TO LS USING INDEX
U 1826, 3EF6,95 :7394      XOR WR[1] WITH LS[BIT0] TO Q,
                        :7395      DT(LONG)&SET.ALU.CC      ;WAS IT WRITTEN IN RIGHT PLACE?
U 1827, CD40,B5 :7396      JMP.IF[EQL] TO [T4.1A]      ;JUMP IF SO
U 1828, 8982,F9 :7397      MOV LS[BIT0] TO WR[0]      ;WRO = RECEIVED DATA
U 1829, B640,15 :7398      JSR [MOV.EX.RE]      ;SET UP FOR ERROR CALL
U 182A, 8870,0C :7399      MISC [SET.CP.ATTN]      ;REPORT ERROR 1
U 182B, 10E0,15 :7400      WAIT.T4.1:
                        :7401      JMP [WAIT.T4.1]      ;WAIT FOR CONSOLE
U 182C, 0982,C4 :7402      JMP [T4.1]      ;LOOP ON ERROR IF ENABLED
U 182D, 0982,54 :7403      JMP [T4.2]      ;GO ON IF NOT
U 182E, 0983,14 :7404
                        :7405
T4.1A:     JSR [TEST.PREV]      ;IS CURRENT ERROR PREVIOUS ERROR?
U 182F, 0871,6C :7406      JMP [T4.1]      ;YES - LOOP
U 1830, 0982,54 :7407
                        :7408
T4.2:     MOV LS[T12] TO WR[0]
U 1831, 3618,15 :7409      MOV WR[0] TO LS[#1]      ;RESTORE LOCATION
U 1832, 3E40,15 :7410      JSR [INC.EN]      ;ERROR NUMBER = 2
U 1833, 8870,5C :7411      MOV LS[#4] TO WR[0]
U 1834, 3644,15 :7412      MOV WR[0] TO LS[OS]      ;OS = 4
U 1835, 3EF8,15 :7413      MOV WR[0] TO LS[T11]     ;T11 WILL BE A SHIFT COUNTER
U 1836, 3E16,15 :7414      LOOP.T4.2:
                        :7415      JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC
U 1837, 8870,9C :7416      MOV WR[1] TO LS[CONTROL.OS] ;WRITE PATTERN TO LS USING INDEX
U 1838, 3EF0,95 :7417      XOR WR[1] WITH LS[ADDRESS.DATA] TO Q,
                        :7418      DT(LONG)&SET.ALU.CC      ;WAS IT WRITTEN IN RIGHT PLACE?
U 1839, 4D88,B5 :7419      JMP.IF[EQL] TO [T4.2A]      ;JUMP IF SO
U 183A, 0984,19 :7420      MOV LS[ADDRESS.DATA] TO WR[0] ;WRO = RECEIVED DATA
U 183B, 3688,15 :7421      JSR [MOV.EX.RE]      ;SET UP FOR ERROR CALL
U 183C, 8870,0C :7422      MISC [SET.CP.ATTN]      ;REPORT ERROR 2
U 183D, 10E0,15 :7423      WAIT.T4.2:
                        :7424      JMP [WAIT.T4.2]      ;WAIT FOR CONSOLE
U 183E, 0983,E4 :7425      JMP [LOOP.T4.2]      ;LOOP ON ERROR IF ENABLED
U 183F, 0983,74 :7426      JMP [T4.3.0]      ;GO ON IF NOT
U 1840, 0984,34 :7427
                        :7428
T4.2A:     
```

U 1841, 0871.6C	:7429	JSR [TEST.PREV]	:IS CURRENT ERROR PREVIOUS ERROR?
U 1842, 0983.14	:7430	JMP [T4.2]	:YES - LOOP
U 1843, 3640.95	:7431		
U 1844, 8870.5C	:7432	T4.3.0: MOV LS[BIT0] TO WR[1]	:INIT SHIFT PATTERN
	:7433	JSR [INC.EN]	:ERROR NUMBER = 3
U 1845, 2F80.15	:7434		
U 1846, BEA0.15	:7435	T4.3: CLR WR[0]	
	:7436	MOV WR[0] TO LS[PREVIOUS.ERROR]	:CLEAR PREVIOUS ERROR
U 1847, 8870.9C	:7437	LOOP.T4.3: JSR [ISSUE.SS]	:ISSUE SCOPE SYNC
U 1848, BFF6.95	:7438	MOV WR[1] TO LS[USER.INDEX(4-0)]	:WRITE PATTERN TO LS
U 1849, B7F6.15	:7439	MOV LS[USER.INDEX(4-0)] TO WR[0]	:WRO = RECEIVED DATA
	:7440	XOR WR[1] WITH WR[0] TO Q,	
U 184A, AB82.35	:7441	DT(LONG)&SET.ALU.CC	:WAS IT WRITTEN?
U 184B, 8985.19	:7442	JMP.IF[EQL] TO [T4.3B]	:GO ON IF SO
U 184C, 8870.0C	:7443	JSR [MOV.EX.RE]	:SET UP FOR ERROR CALL
U 184D, 10E0.15	:7444	MISC [SET.CP.ATTN]	:REPORT ERROR 3
	:7445		
U 184E, 8984.E4	:7446	WAIT.T4.3A: JMP [WAIT.T4.3A]	:WAIT FOR CONSOLE
U 184F, 8984.74	:7447	JMP [LOOP.T4.3]	:LOOP ON ERROR IF ENABLED
U 1850, 8985.34	:7448	JMP [NOLoop.T4.3A]	:GO ON OF NOT
	:7449		
U 1851, 0871.6C	:7450	T4.3B: JSR [TEST.PREV]	:IS CURRENT ERROR PREVIOUS ERROR?
U 1852, 8984.74	:7451	JMP [LOOP.T4.3]	:YES - LOOP
	:7452		
U 1853, CD10.B5	:7453	NOLoop.T4.3A: XOR WR[1] WITH LS[T8] TO Q,	
U 1854, 8985.79	:7454	DT(LONG)&SET.ALU.CC	:IS WR1 = BIT31 YET?
U 1855, A3D0.95	:7455	JMP.IF[EQL] TO [T4.3C]	:IF SO GO TEST NEXT LOCATION
U 1856, 0984.54	:7456	ASHL WR[1]	:SHIFT PATTERN
	:7457	JMP [T4.3]	:TEST NEXT BIT
	:7458		
U 1857, 3640.95	:7459	T4.3C: MOV LS[#1] TO WR[1]	:RESET PATTERN
U 1858, B6F8.15	:7460	MOV LS[OS] TO WR[0]	:GET OS
U 1859, 2040.15	:7461	INC WR[0]	:INCREMENT IT
U 185A, 3EF8.15	:7462	MOV WR[0] TO LS[OS]	:PUT BACK IN OS
U 185B, B616.15	:7463	MOV LS[T11] TO WR[0]	:GET SHIFT COUNTER
	:7464	XOR WR[0] WITH LS[T8] TO Q,	
U 185C, 4D10.35	:7465	DT(LONG)&SET.ALU.CC	:IS OS = 32. ?
U 185D, 23D0.15	:7466	ASHL WR[0]	:SHIFT COUNTER
U 185E, 3E16.15	:7467	MOV WR[0] TO LS[T11]	:PUT SHIFTED VALUE BACK
U 185F, 5B00.09	:7468	SKIP.IF[EQL]	:SKIP IF SO
U 1860, 0984.54	:7469	JMP [T4.3]	:TEST NEXT LOCATION
	:7470		
U 1861, 8870.5C	:7471	:	
U 1862, 2F80.15	:7472	JSR [INC.EN]	:INC ERROR NUMBER TO 4
U 1863, 3EF8.15	:7473	CLR WR[0]	
U 1864, B618.95	:7474	MOV WR[0] TO LS[OS]	:CLEAR THE OS REGISTER
U 1865, BE16.95	:7475	MOV LS[T12] TO WR[1]	:WR1 = FLOAT PATTERN = #00000001 INITIALLY
	:7476	MOV WR[1] TO LS[T11]	:RESET THE SHIFT COUNTER TO ONE
	:7477		
U 1866, 2F80.15	:7478	T4.4: CLR WR[0]	
U 1867, BEA0.15	:7479	MOV WR[0] TO LS[PREVIOUS.ERROR]	:CLEAR PREVIOUS ERROR
	:7480		
U 1868, 8870.9C	:7481	LOOP.T4.4: JSR [ISSUE.SS]	:ISSUE SCOPE SYNC
U 1869, BFF0.95	:7482	MOV WR[1] TO LS[XGPR.OS]	:WRITE PATTERN TO LS
U 186A, B7F0.15	:7483	MOV LS[XGPR.OS] TO WR[0]	:WRO = RECEIVED DATA

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

TEST 4 - Index Mode 3-MAR-82 E 14
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 4 - Index Mode Into LS At Speed test (M8390 CPU Module) Fiche 1 Frame E14 Sequence 173
Rev 01.00 Page 167

```

:7484      XOR WR[1] WITH WR[0] TO Q,
U 186B, AB82.35 :7485      DT(LONG)&SET.ALU.CC      ;WAS IT WRITTEN?
U 186C, 0987.29 :7486      JMP.IF[EQ] TO [T4.4B]      ;GO ON IF SO
U 186D, 8870.0C :7487      JSR [MOV.EX.RE]      ;SET UP FOR ERROR CALL.
U 186E, 10E0.15 :7488      MISC [SET.CP.ATTN]      ;REPORT ERROR 4
:7489      WAIT.T4.4A:
U 186F, 8986.F4 :7490      JMP [WAIT.T4.4A]      ;WAIT FOR CONSOLE
U 1870, 0986.84 :7491      JMP [LOOP.T4.4]      ;LOOP ON ERROR IF ENABLED
U 1871, 8987.44 :7492      JMP [NOLoop.T4.4A]      ;GO ON OF NOT
:7493      T4.4B:
U 1872, 0871.6C :7494      JSR [TEST.PREV]      ;IS CURRENT ERROR PREVIOUS ERROR?
U 1873, 0986.84 :7495      JMP [LOOP.T4.4]      ;YES - LOOP
:7496      NOLoop.T4.4A:
U 1874, CD10.B5 :7497      XOR WR[1] WITH LS[T8] TO Q,
U 1875, 0987.89 :7498      DT(LONG)&SET.ALU.CC      ;IS WR1 = BIT31 YET?
U 1876, A3D0.95 :7499      JMP.IF[EQ] TO [T4.4C]      ;IF SO GO TEST NEXT LOCATION
U 1877, 8986.64 :7500      ASHL WR[1]      ;SHIFT PATTERN
:7501      JMP [T4.4]      ;TEST NEXT BIT
:7502      T4.4C:
U 1878, 3640.95 :7503      MOV LS[#1] TO WR[1]      ;RESET PATTERN
U 1879, B6F8.15 :7504      MOV LS[OS] TO WR[0]      ;GET OS
U 187A, 2040.15 :7505      INC WR[0]      ;INCREMENT IT
U 187B, 3EF8.15 :7506      MOV WR[0] TO LS[OS]      ;INCREMENTED OS REGISTER
U 187C, B616.15 :7507      MOV LS[T11] TO WR[0]      ;GET SHIFT COUNTER
:7508      XOR WR[0] WITH LS[T10] TO Q,
U 187D, CD14.35 :7509      DT(LONG)&SET.ALU.CC      ;IS OS = 16. ?
U 187E, 23D0.15 :7510      ASHL WR[0]      ;SHIFT IT
U 187F, 3E16.15 :7511      MOV WR[0] TO LS[T11]      ;RESTORE THE SHIFT COUNTER
U 1880, 5B00.09 :7512      SKIP.IF[EQ]      ;SKIP IF SO
U 1881, 8986.64 :7513      JMP [T4.4]      ;TEST NEXT LOCATION
:7514      :
U 1882, 8870.5C :7515      JSR [INC.EN]      ;INC ERROR NUMBER TO 5
U 1883, 2F80.15 :7516      CLR WR[0]
U 1884, 3EF8.15 :7517      MOV WR[0] TO LS[OS]      ;CLEAR THE OS REGISTER
U 1885, BE16.95 :7518      MOV WR[1] TO LS[T11]      ;CLEAR SHIFT COUNTER TO ONE
:7519      :
:7520      ;WRITE BACKGROUND OF ZEROS
:7521      T4.WRITE.ZERO:
U 1886, 3FF6.15 :7522      MOV WR[0] TO LS[USER.INDEX(4-0)];WRITE ZERO
U 1887, 36F8.95 :7523      MOV LS[OS] TO WR[1]
U 1888, 2042.95 :7524      INC WR[1]
U 1889, BEF8.95 :7525      MOV WR[1] TO LS[OS]      ;INCREMENT INDEX
U 188A, 3616.95 :7526      MOV LS[T11] TO WR[1]      ;GET COUNTER
:7527      XOR WR[1] WITH LS[T8] TO Q,
U 188B, CD10.B5 :7528      DT(LONG)&SET.ALU.CC      ;IS OS = 32. ?
U 188C, A3D0.95 :7529      ASHL WR[1]      ;INCREMENT COUNTER
U 188D, BE16.95 :7530      MOV WR[1] TO LS[T11]      ;AND RESTORE IT IN T11
U 188E, 0988.61 :7531      JMP.IF[NEQ] TO [T4.WRITE.ZERO] ;JUMP IF NOT
U 188F, 3EF8.15 :7532      MOV WR[0] TO LS[OS]      ;CLEAR OS
U 1890, 3640.95 :7533      MOV LS[#1] TO WR[1]
U 1891, BE16.95 :7534      MOV WR[1] TO LS[T11]      ;RESET SHIFT COUNTER
:7535      :
:7536      T4.5:
U 1892, 2F80.15 :7537      CLR WR[0]
U 1893, BEA0.15 :7538      MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR
```



```

:7539 LOOP.T4.5:
U 1894, 8870.9C :7540 JSR [ISSUE.SS] ;ISSUE SCOPE SYNC
U 1895, 2F82.95 :7541 CLR WR[1]
U 1896, B7F6.15 :7542 MOV LS[USER.INDEX(4-0)] TO WR[0];WRO = RECEIVED DATA
:7543 XOR WR[1] WITH WR[0] TO Q,
U 1897, AB82.35 :7544 DT(LONG)&SET.ALU.CC ;IS THE LOCATION CLEAR ?
U 1898, 8989.E9 :7545 JMP.IF[EQL] TO [T4.5A] ;JUMP IF SO
U 1899, 8870.0C :7546 JSR [MOV.EX.RE] ;SET UP FOR ERROR CALL
U 189A, 10E0.15 :7547 MISC [SET.CP.ATTN] ;REPORT ERROR 5
:7548 WAIT.T4.5:
U 189B, 0989.B4 :7549 JMP [WAIT.T4.5] ;WAIT FOR CONSOLE
U 189C, 0989.44 :7550 JMP [LOOP.T4.5] ;LOOP ON ERROR IF ENABLED
U 189D, 898A.04 :7551 JMP [LOOP.T4.5B] ;GO ON IF NOT
:7552 T4.5A:
U 189E, 0871.6C :7553 JSR [TEST.PREV] ;IS CURRENT ERROR PREVIOUS ERROR?
U 189F, 0989.44 :7554 JMP [LOOP.T4.5] ;YES - LOOP
:7555 LOOP.T4.5B:
U 18A0, 8870.9C :7556 JSR [ISSUE.SS] ;ISSUE SCOPE SYNC
U 18A1, 369E.95 :7557 MOV LS[ONES] TO WR[1] ;WR1 = #FFFFFFFF
U 18A2, BFF6.95 :7558 MOV WR[1] TO LS[USER.INDEX(4-0)];WRITE ONES TO LS LOCATION
U 18A3, B7F6.15 :7559 MOV LS[USER.INDEX(4-0)] TO WR[0];WRO = RECEIVED DATA
:7560 XOR WR[1] WITH WR[0] TO Q,
U 18A4, AB82.35 :7561 DT(LONG)&SET.ALU.CC ;WAS IT WRITTEN ?
U 18A5, 898A.B9 :7562 JMP.IF[EQL] TO [T4.5B] ;JUMP IF SO
U 18A6, 8870.0C :7563 JSR [MOV.EX.RE] ;SET UP FOR ERROR CALL
U 18A7, 10E0.15 :7564 MISC [SET.CP.ATTN] ;REPORT ERROR 5
:7565 WAIT.T4.5B:
U 18A8, 098A.84 :7566 JMP [WAIT.T4.5B] ;WAIT FOR CONSOLE
U 18A9, 898A.04 :7567 JMP [LOOP.T4.5B] ;LOOP ON ERROR IF ENABLED
U 18AA, 098A.D4 :7568 JMP [NOLoop.T4.5] ;GO ON IF NOT
:7569 T4.5B:
U 18AB, 0871.6C :7570 JSR [TEST.PREV] ;IS CURRENT ERROR PREVIOUS ERROR ?
U 18AC, 898A.04 :7571 JMP [LOOP.T4.5B] ;YES - LOOP
:7572 NOLoop.T4.5:
U 18AD, B6F8.15 :7573 MOV LS[OS] TO WR[0]
U 18AE, 2040.15 :7574 INC WR[0] ;INCREMENT OS REGISTER
U 18AF, 3EF8.15 :7575 MOV WR[0] TO LS[OS] ;GET COUNTER
U 18B0, B616.15 :7576 MOV LS[T11] TO WR[0]
:7577 XOR WR[0] WITH LS[T8] TO Q,
U 18B1, 4D10.35 :7578 DT(LONG)&SET.ALU.CC ;IS OS = 32 ?
U 18B2, 23D0.15 :7579 ASHL WR[0] ;INCREMENT COUNTER
U 18B3, 3E16.15 :7580 MOV WR[0] TO LS[T11] ;RESTORE COUNTER
U 18B4, 5B00.09 :7581 SKIP.IF[EQL] ;SKIP IF SO
U 18B5, 0989.24 :7582 JMP [T4.5] ;TEST NEXT LOCATION
U 18B6, 8870.5C :7583 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 6
U 18B7, 2F80.15 :7584 CLR WR[0]
U 18B8, 3EF8.15 :7585 MOV WR[0] TO LS[OS] ;CLEAR OS
U 18B9, B640.15 :7586 MOV LS[#1] TO WR[0]
U 18BA, 3E16.15 :7587 MOV WR[0] TO LS[T11] ;CLEAR SHIFT COUNTER
:7588 T4.6:
U 18BB, 2F80.15 :7589 CLR WR[0]
U 18BC, BEA0.15 :7590 MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR
:7591 LOOP.T4.6:
U 18BD, 8870.9C :7592 JSR [ISSUE.SS] ;ISSUE SCOPE SYNC
U 18BE, 369E.95 :7593 MOV LS[ONES] TO WR[1] ;WR1 = #FFFFFFFF
    
```

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

G 14
TEST 4 - Index Mode 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module Rev 01.00
TEST 4 - Index Mode Into LS At Speed test (M8390 CPU Module) Fiche 1 Frame G14 Sequence 175 Page 169

```
U 18BF, B7F6,15 :7594      MOV LS[USER.INDEX(4-0)] TO WR[0];WRO = RECEIVED DATA
:7595      XOR WR[1] WITH WR[0] TO Q,
U 18C0, AB82,35 :7596      DT(LONG)&SET.ALU.CC      :IS THE LOCATION ALL ONES ?
U 18C1, 898C,79 :7597      JMP.IF[EQL] TO [T4.6A]      :JUMP IF SO
U 18C2, 8870,0C :7598      JSR [MOV.EX.RE]      :SET UP FOR ERROR CALL
U 18C3, 10E0,15 :7599      MISC [SET.CP.ATTN]      :REPORT ERROR 5
:7600      WAIT.T4.6:
U 18C4, 098C,44 :7601      JMP [WAIT.T4.6]      :WAIT FOR CONSOLE
U 18C5, 898B,D4 :7602      JMP [LOOP.T4.6]      :LOOP ON ERROR IF ENABLED
U 18C6, 898C,94 :7603      JMP [LOOP.T4.6B]      :GO ON IF NOT
:7604      T4.6A:
U 18C7, 0871,6C :7605      JSR [TEST.PREV]      :IS CURRENT ERROR PREVIOUS ERROR?
U 18C8, 898B,D4 :7606      JMP [LOOP.T4.6]      :YES - LOOP
:7607      LOOP.T4.6B:
U 18C9, 8870,9C :7608      JSR [ISSUE.SS]      :ISSUE SCOPE SYNC
U 18CA, 2F82,95 :7609      CLR WR[1]
U 18CB, BFF6,95 :7610      MOV WR[1] TO LS[USER.INDEX(4-0)];CLEAR LS LOCATION
U 18CC, B7F6,15 :7611      MOV LS[USER.INDEX(4-0)] TO WR[0];WRO = RECEIVED DATA
:7612      XOR WR[1] WITH WR[0] TO Q,
U 18CD, AB82,35 :7613      DT(LONG)&SET.ALU.CC      :WAS IT CLEARED ?
U 18CE, 098D,49 :7614      JMP.IF[EQL] TO [T4.6B]      :JUMP IF SO
U 18CF, 8870,0C :7615      JSR [MOV.EX.RE]      :SET UP FOR ERROR CALL
U 18D0, 10E0,15 :7616      MISC [SET.CP.ATTN]      :REPORT ERROR 5
:7617      WAIT.T4.6B:
U 18D1, 898D,14 :7618      JMP [WAIT.T4.6B]      :WAIT FOR CONSOLE
U 18D2, 898C,94 :7619      JMP [LOOP.T4.6B]      :LOOP ON ERROR IF ENABLED
U 18D3, 098D,64 :7620      JMP [NOLOOP.T4.6]      :GO ON IF NOT
:7621      T4.6B:
U 18D4, 0871,6C :7622      JSR [TEST.PREV]      :IS CURRENT ERROR PREVIOUS ERROR ?
U 18D5, 898C,94 :7623      JMP [LOOP.T4.6B]      :YES - LOOP
:7624      NOLOOP.T4.6:
U 18D6, B6F8,15 :7625      MOV LS[OS] TO WR[0]
U 18D7, 2040,15 :7626      INC WR[0]
U 18D8, 3EF8,15 :7627      MOV WR[0] TO LS[OS]      :INCREMENT OS REGISTER
U 18D9, B616,15 :7628      MOV LS[T11] TO WR[0]      :GET COUNTER
:7629      XOR WR[0] WITH LS[T8] TO Q,
U 18DA, 4D10,35 :7630      DT(LONG)&SET.ALU.CC      :IS OS = 32 ?
U 18DB, 23D0,15 :7631      ASHL WR[0]      :INCREMENT COUNTER
U 18DC, 3E16,15 :7632      MOV WR[0] TO LS[T11]      :RESTORE COUNTER
U 18DD, 5B00,09 :7633      SKIP.IF[EQL]      :SKIP IF SO
U 18DE, 898B,B4 :7634      JMP [T4.6]      :TEST NEXT LOCATION
:7635      END.T4:
```


Page "TEST 5 - BIS Instruction (M8390 CPU Module)"

Test Description:

This test checks the BIS instruction that is used by the error routine. The BIS instruction used is BIS LS[] to WR[] and the 2901 is set up for the following mode: SRC=D,A DST=WRITE.B.F FUNC=R OR S (D OR A) CIN=CIN. The data path is as follows: The contents of LS is put on the D BUS and 'OR'ed with the WR specified by the u-instruction. The result is written back into the same WR and the result is also put on the Y BUS. The result is compared with a location in LS using the previously tested XOR WR WITH LS TO Q instruction.

Assumptions:

It is assumed that the XOR WR WITH LS TO Q instruction is working correctly because it was tested in a previous test. It is also assumed that the LS is previously loaded correctly with constants (the LS is loaded when the overlay is loaded and also at the end of the LS test).

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Try BIS instruction with data of 0 from LS and 0 in the WR. Check result for 0 with skip on ALU not equal 0 in SCTL field.
- 3) Check BIS with 0 from LS and all ones in the WR. Check result for non-zero with skip on ALU=0 in SCTL field.
- 4) Using XOR WR WITH LS TO Q, check result for all ones.
- 5) Check BIS with all ones from LS and 0 in the WR. Check result for all ones.
- 6) Check BIS with all ones from LS and all ones in the WR. Check result for all ones again.
- 7) Using a shifting 1 pattern from LS, check that each bit can be set separately. Repeat until all 32 bits have been tested. This check is done using index mode and the OS reg.

Errors:

- Error 1 - BIS LS WITH WR failed with data of 0 and 0.
Error 2 - BIS LS WITH WR failed with data of 0 and all ones. SCTL=skip on ALU=0 skipped.
Error 3 - BIS LS WITH WR failed with data of 0 and all ones. Failed to store all ones in the WR.
Error 4 - BIS LS WITH WR failed with data of all ones and 0.
Error 5 - BIS LS WITH WR failed with data of all ones and all ones.
Error 6 - BIS LS WITH WR failed with data of shifting ones and 0.
Error 7 - BIS LS WITH WR failed with data from LS location in range 0-1F.
Error 8 - BIS LS WITH WR failed with data from LS location in range 40-4F.

Debug:

- Error 1 - This indicates a basic failure of the 2901 or it's control

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 5 - BIS Instru 3-14
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 5 - BIS Instruction (M8390 CPU Module)

Fiche 1 Frame I14

Sequence 177
Rev 01.00 Page 171

```
:7691 : lines. The most likely problem is with the control lines or drivers.
:7692 : Referring to the chart at the beginning of this listing and this test
:7693 : description, check all control lines going to the 2901. If there is
:7694 : a problem here, check the output directly out of the DEST CTL ROM and
:7695 : the SRC.ALU CTL PAL. The inputs to these ICs, from CSR 22 through CSR
:7696 : 14 should be 1 0001 1101(B) respectively. This check should be made
:7697 : with the CPU single stepped to the BIS instruction. If the control
:7698 : lines are OK then suspect the 2901 itself. The error in this case
:7699 : will most likely be with one bit or a group of 4 bits.
:7700 :
:7701 : Error 2 - If this is the first failure, the 2901 itself is the most
:7702 : likely problem. However if all bits are failing, the control signals
:7703 : should be checked out as described in error 1 above.
:7704 :
:7705 : Error 3 - Same as error 2. A possible problem is that the write back
:7706 : to the WR is not functioning or the DEST CTL is incorrect.
:7707 :
:7708 : Error 4 - Same as error 2.
:7709 :
:7710 : Error 5 - Same as error 2.
:7711 :
:7712 : Error 6 - This error indicates either a problem with the 2901 or the
:7713 : DEST CTL ROM or SRC.ALU CTL PAL. The input to these IC's differs by
:7714 : one bit from the previous BIS instructions since the shifting one
:7715 : pattern uses index mode into the LS. CSR 22 through CSR 14 going to
:7716 : the ROM and PAL is 1 0001 1111(B) respectively. Since a new location
:7717 : in the ROM is being addressed and a new pattern is going to the PAL,
:7718 : these areas are suspect. If all bits are not failing, suspect an
:7719 : internal fault in the respective 2901.
:7720 :
:7721 : Error 7 - Same as error 6 except that CSR 22 through 14 going to the
:7722 : ROM and PAL should be 1 0001 1100.
:7723 :
:7724 : Error 8 - Same as error 6 except that CSR 22 through 14 going to the
:7725 : ROM and PAL should be 1 0001 1110.
:7726 :
:7727 :
:7728 :
:7729 :
:7730 :
:7731 :
:7732 :
:7733 :
:7734 :
:7735 :
:7736 :
:7737 :
:7738 :
:7739 :
:7740 :
:7741 :
:7742 :
:7743 :
:7744 :
:7745 :
```

U 18DF, B65E,15 :7728
U 18E0, 3E80,15 :7729
U 18E1, 10E0,15 :7730
U 18E2, 898E,24 :7731
U 18E3, 2F80,15 :7732
U 18E4, BEA0,15 :7733
U 18E5, 3E8A,15 :7734
U 18E6, 2040,15 :7735
U 18E7, BE82,15 :7736
U 18E8, 2040,15 :7737
U 18E9, 3E8C,15 :7738
U 18EA, 367E,15 :7739
U 18EB, BE14,15 :7740
U 18EC, B63E,15 :7741
U 18ED, 3E80,15 :7742
:7743
:7744
:7745

T.5:--
WAIT.T5.0:

```
MOV LS[BEGIN.TEST] TO WR[0] :SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] :SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN] :BIT15 INDICATES START OF TEST TO CONSOLE
:ISSUE CPU ATTN TO CONSOLE
JMP [WAIT.T5.0] :LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR WR[0] :SET WRO TO 0
MOV WR[0] TO LS[PREVIOUS.ERROR] :CLEAR PREVIOUS ERROR NUMBER
MOV WR[0] TO LS[ERROR.MASK] :CLEAR ERROR MASK.
INC WR[0] :SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER] :SET ERROR NUMBER TO FIRST ERROR
INC WR[0] :SET WRO TO 2
MOV WR[0] TO LS[MODULE.NUM] :SET UP MODULE CODE FOR CPU MODULE
MOV LS[BIT31] TO WR[0] :SET WRO TO BIT 31
MOV WR[0] TO LS[T10] :SET T10 TO BIT 31
MOV LS[ERR.CON] TO WR[0] :LOAD WRO WITH A CONSTANT THAT SETS BITS 8 & 10.
MOV WR[0] TO LS[CONTROL.STATUS] :LOAD CONTROL AND STATUS WORD. BIT 8
: SET INDICATES AN ERROR, BIT 10 SET
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) TEST 5 - BIS Instru 3-MAR-82
TEST 5 - BIS Instruction (M8390 CPU Module)

Fiche 1 Frame J14

Sequence 178
Rev 01.00 Page 172

```
:7746  
:7747  
:7748  
U 18EE, 2F80,15 :7749  
U 18EF, 8870,9C :7750  
:7751  
U 18F0, C79C,35 :7752  
U 18F1, DB00,01 :7753  
U 18F2, 898F,94 :7754  
U 18F3, 2F82,95 :7755  
U 18F4, 8870,0C :7756  
U 18F5, 10E0,15 :7757  
:7758  
U 18F6, 898F,64 :7759  
U 18F7, 898E,E4 :7760  
U 18F8, 898F,C4 :7761  
:7762  
U 18F9, 36A0,15 :7763  
:7764  
U 18FA, CD82,35 :7765  
U 18FB, 098E,E9 :7766  
:7767  
:7768  
U 18FC, 3642,15 :7769  
U 18FD, BE82,15 :7770  
U 18FE, B69E,15 :7771  
U 18FF, 8870,9C :7772  
:7773  
U 1900, C79C,35 :7774  
:7775  
U 1901, BE10,09 :7776  
U 1902, 0990,94 :7777  
U 1903, 369E,95 :7778  
U 1904, 8870,0C :7779  
U 1905, 10E0,15 :7780  
:7781  
U 1906, 0990,64 :7782  
U 1907, 898F,C4 :7783  
U 1908, 0990,C4 :7784  
:7785  
U 1909, 36A0,15 :7786  
:7787  
U 190A, CD82,35 :7788  
U 190B, 098F,C9 :7789  
:7790  
U 190C, 8870,5C :7791  
U 190D, B610,15 :7792  
:7793  
U 190E, 4D9E,35 :7794  
U 190F, DB00,01 :7795  
U 1910, 0991,74 :7796  
U 1911, 369E,95 :7797  
U 1912, 8870,0C :7798  
U 1913, 10E0,15 :7799  
:7800  
LOOP.T5.1:  
CLR WR[0]  
JSR [ISSUE.SS]  
BIS LS[ZERO] TO WR[0],  
DT(LONG)&SET.ALU.CC  
SKIP,IF[NEQ]  
JMP [T5.2]  
CLR WR[1]  
JSR [MOV.EX.RE]  
MISC [SET.CP.ATTN]  
WAIT.T5.1:  
JMP [WAIT.T5.1]  
JMP [LOOP.T5.1]  
JMP [NOLOOP.T5.1]  
T5.2:  
MOV LS[PREVIOUS.ERROR] TO WR[0]  
XOR WR[0] WITH LS[ERROR.NUMBER]  
DT(LONG)&SET.ALU.CC  
JMP,IF[EQL] TO [LOOP.T5.1]  
NOLOOP.T5.1:  
LOOP.T5.2:  
MOV LS[#2] TO WR[0]  
MOV WR[0] TO LS[ERROR.NUMBER]  
MOV LS[ONES] TO WR[0]  
JSR [ISSUE.SS]  
BIS LS[ZERO] TO WR[0],  
DT(LONG)&SET.ALU.CC  
MOV WR[0] TO LS[T8],  
SKIP,IF[EQL]  
JMP [T5.3]  
MOV LS[ONES] TO WR[1]  
JSR [MOV.EX.RE]  
MISC [SET.CP.ATTN]  
WAIT.T5.2:  
JMP [WAIT.T5.2]  
JMP [LOOP.T5.2]  
JMP [NOLOOP.T5.2]  
T5.3:  
MOV LS[PREVIOUS.ERROR] TO WR[0]  
XOR WR[0] WITH LS[ERROR.NUMBER]  
DT(LONG)&SET.ALU.CC  
JMP,IF[EQL] TO [LOOP.T5.2]  
NOLOOP.T5.2:  
JSR [INC.EN]  
MOV LS[T8] TO WR[0]  
XOR WR[0] WITH LS[ONES] TO Q,  
DT(LONG)&SET.ALU.CC  
SKIP,IF[NEQ]  
JMP [T5.4]  
MOV LS[ONES] TO WR[1]  
JSR [MOV.EX.RE]  
MISC [SET.CP.ATTN]  
WAIT.T5.3:  
: INDICATES CONSOLE CONTROLS ERROR  
: LOOPING.  
: CLEAR OUT WORKING REGISTER  
: ISSUE SCOPE SYNC  
: TRY BIS INS. WITH DATA OF ZERO AND WR=0  
: SKIP NEXT INSTRUCTION IF A BIT WAS SET  
: GOTO NEXT SECTION  
: PUT EXPECTED DATA IN WR1  
: SET UP EXPECTED AND RECEIVED DATA  
: REPORT ERROR 1 TO CONSOLE  
: WAIT FOR CONSOLE TO RESPOND  
: LOOP ON ERROR IF ENABLE  
: GOTO NEXT TEST IF NO LOOPING  
: GET PREVIOUS ERROR NUMBER  
TO Q, ;CURRENT ERROR?  
: LOOP ON ERROR IF SO  
: GET DATA OF 2  
: AND SET ERROR NUMBER TO IT  
: SET ALL BITS IN WR0  
: ISSUE SCOPE SYNC  
: TRY BIS INS. WITH DATA OF ZERO AND ALL BITS SET IN WR  
: SAVE DATA  
: AND SKIP NEXT INSTRUCTION IF REGISTER WAS CLEARED  
: GO TO NEXT SECTION  
: PUT EXPECTED DATA IN WR1  
: SET UP EXPECTED AND RECEIVED DATA  
: REPORT ERROR 2 TO CONSOLE  
: WAIT FOR CONSOLE TO RESPOND  
: LOOP ON ERROR IF ENABLE  
: GOTO NEXT TEST IF NO LOOPING  
: GET PREVIOUS ERROR NUMBER  
TO Q, ;CURRENT ERROR?  
: LOOP ON ERROR IF SO  
: INCREMENT ERROR NUMBER TO 3  
: GET SAVED DATA  
: ARE ALL BITS STILL SET?  
: SKIP NEXT INSTRUCTION IF ALL BITS WERE NOT STILL SET  
: PUT EXPECTED DATA IN WR1  
: SET UP EXPECTED AND RECEIVED DATA  
: REPORT ERROR 3 TO CONSOLE
```


ZZ-ENKCA-1.0 : ENKCA.MIC
 : ENKCA.MCR
 : ENKCA.MIC

K 14
 TEST 5 - BIS Instru 3-MAR-82
 MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
 TEST 5 - BIS Instruction (M8390 CPU Module)

Fiche 1 Frame K14
 Sequence 179
 Rev 01.00 Page 173

U 1914, 0991.44	:7801	JMP [WAIT.T5.3]	:WAIT FOR CONSOLE TO RESPOND
U 1915, 898F.C4	:7802	JMP [LOOP.T5.2]	:LOOP ON ERRO IF ENABLED
U 1916, 8991.A4	:7803	JMP [NOLOOP.T5.3]	:GOTO NEXT TEST IF NO LOOPING
	:7804		
U 1917, 36A0.15	:7805	T5.4: MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR
	:7806	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1918, CD82.35	:7807	DT(LONG)&SET.ALU.CC	
U 1919, 098F.C9	:7808	JMP.IF[EQL] TO [LOOP.T5.2]	:LOOP ON ERROR IF SO
	:7809	NOLOOP.T5.3:	
U 191A, 8870.5C	:7810	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 4
	:7811	LOOP.T5.4:	
U 191B, 2F80.15	:7812	CLR WR[0]	:CLEAR ALL BITS IN WRO
U 191C, 8870.9C	:7813	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC
U 191D, C79E.15	:7814	BIS LS[ONES] TO WR[0]	:TRY BIS INS. WITH DATA OF ONES AND CLEAR WR
	:7815	XOR WR[0] WITH LS[ONES] TO Q,	:WERE ALL BITS SET IN WRO?
U 191E, 4D9E.35	:7816	DT(LONG)&SET.ALU.CC	
U 191F, DB00.01	:7817	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF ALL BITS WERE NOT SET
U 1920, 0992.74	:7818	JMP [T5.5]	:GOTO NEXT SECTION
U 1921, 369E.95	:7819	MOV LS[ONES] TO WR[1]	:PUT EXPECTED DATA IN WR1
U 1922, 8870.0C	:7820	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 1923, 10E0.15	:7821	MISC [SET.CP.ATTN]	:REPORT ERROR 4 TO CONSOLE
	:7822	WAIT.T5.4:	
U 1924, 0992.44	:7823	JMP [WAIT.T5.4]	:WAIT FOR CONSOLE TO RESPOND
U 1925, 0991.B4	:7824	JMP [LOOP.T5.4]	:LOOP ON ERROR IF ENABLED
U 1926, 8992.A4	:7825	JMP [NOLOOP.T5.4]	:GOTO NEXT TEST IF NO LOOPING
	:7826		
U 1927, 36A0.15	:7827	T5.5: MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR NUMBER
	:7828	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1928, CD82.35	:7829	DT(LONG)&SET.ALU.CC	
U 1929, 8991.B9	:7830	JMP.IF[EQL] TO [LOOP.T5.4]	:LOOP ON ERROR IF SO
	:7831	NOLOOP.T5.4:	
U 192A, 8870.5C	:7832	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 5
	:7833	LOOP.T5.5:	
U 192B, B69E.15	:7834	MOV LS[ONES] TO WR[0]	:SET ALL BITS IN WRO
U 192C, 8870.9C	:7835	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC
U 192D, C79E.15	:7836	BIS LS[ONES] TO WR[0]	:TRY BIS INS. WITH ONES IN BOTH DATA AND WR
	:7837	XOR WR[0] WITH LS[ONES] TO Q,	:ARE ALL BITS STILL SET?
U 192E, 4D9E.35	:7838	DT(LONG)&SET.ALU.CC	
U 192F, DB00.01	:7839	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF NOT
U 1930, 8993.74	:7840	JMP [T5.6.0]	:GOTO NEXT SECTION
U 1931, 369E.95	:7841	MOV LS[ONES] TO WR[1]	:PUT EXPECTED DATA IN WR1
U 1932, 8870.0C	:7842	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 1933, 10E0.15	:7843	MISC [SET.CP.ATTN]	:REPORT ERROR 5 TO CONSOLE
	:7844	WAIT.T5.5:	
U 1934, 8993.44	:7845	JMP [WAIT.T5.5]	:WAIT FOR CONSOLE TO RESPOND
U 1935, 0992.B4	:7846	JMP [LOOP.T5.5]	:LOOP ON ERROR IF ENABLED
U 1936, 0993.A4	:7847	JMP [NOLOOP.T5.5]	:GOTO NEXT TEST IF NO LOOPING
	:7848		
U 1937, 36A0.15	:7849	T5.6.0: MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR NUMBER
	:7850	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1938, CD82.35	:7851	DT(LONG)&SET.ALU.CC	
U 1939, 8992.B9	:7852	JMP.IF[EQL] TO [LOOP.T5.5]	:LOOP ON ERROR IF SO
	:7853	NOLOOP.T5.5:	
U 193A, 8870.5C	:7854	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 6
U 193B, 2F80.15	:7855	CLR WR[0]	:SET WRO TO 0

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

MICRO2 1L(03) TEST 5 - BIS Instru 3-MAR-82
3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 5 - BIS Instruction (M8390 CPU Module)

L 14
Fiche 1 Frame L14

Sequence 180
Rev 01.00 Page 174

```
U 193C, 3EF8,15 :7856      MOV WR[0] TO LS[OS]           ;CLEAR THE OS REGISTER (FOR INDEXING)
U 193D, B640,15 :7857      MOV LS[11] TO WR[0]           ;RESET COUNTER
U 193E, 3E16,15 :7858      MOV WR[0] TO LS[T11]
:7859
T5.6:      CLR WR[0]
U 193F, 2F80,15 :7860      MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR
U 1940, BEA0,15 :7861
:7862
LOOP.T5.6: CLR WR[0]           ;CLEAR WRO
:7863      JSR [ISSUE.SS]         ;ISSUE SCOPE SYNC
U 1941, 2F80,15 :7864
U 1942, 8870,9C :7865      BIS LS[SHIFT.OS(4-0)] TO WR[0] ;TRY BIS USING ONLY ONE BIT AT A TIME USING INDEX MODE
U 1943, 47F6,15 :7866      XOR WR[0] WITH LS[SHIFT.OS(4-0)] TO Q, ;IS THE BIT SET?
:7867      DT(LONG)&SET.ALU.CC
U 1944, CDF6,35 :7868      SKIP.IF[NEQ]           ;SKIP NEXT INSTRUCTION IF NOT
U 1945, DB00,01 :7869      JMP [T5.6A]             ;SET UP FOR NEXT BIT
U 1946, 0994,D4 :7870      MOV LS[SHIFT.OS(4-0)] TO WR[1] ;PUT EXPECTED DATA IN WR1
U 1947, B6F6,95 :7871      JSR [MOV.EX.RE]         ;SET UP EXPECTED AND RECEIVED DATA
U 1948, 8870,0C :7872      MISC [SET.CP.ATTN]       ;REPORT ERROR 6 TO CONSOLE
U 1949, 10E0,15 :7873
:7874
WAIT.T5.6: JMP [WAIT.T5.6]       ;WAIT FOR CONSOLE TO RESPOND
U 194A, 8994,A4 :7875      JMP [LOOP.T5.6]         ;LOOP ON ERROR IF ENABLED
U 194B, 0994,14 :7876      JMP [NOLOOP.T5.6]        ;GOTO NEXT TEST IF NO LOOPING
U 194C, 0995,04 :7877
:7878
T5.6A:     MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
U 194D, 36A0,15 :7879      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
:7880      DT(LONG)&SET.ALU.CC
U 194E, CD82,35 :7881      JMP.IF[EQL] TO [LOOP.T5.6] ;LOOP ON ERROR IF SO
U 194F, 8994,19 :7882
:7883
NOLOOP.T5.6: MOV LS[OS] TO WR[0] ;GET INDEX
U 1950, B6F8,15 :7884      INC WR[0]           ;AND INCREMENT
U 1951, 2040,15 :7885      MOV WR[0] TO LS[OS]         ;IT FOR NEXT BIT
U 1952, 3EF8,15 :7886      MOV LS[T11] TO WR[0]       ;INCREMENT COUNTER
U 1953, B616,15 :7887      XOR WR[0] WITH LS[T10] TO Q, ;IS INDEX =32?
:7888      DT(LONG)&SET.ALU.CC
U 1954, CD14,35 :7889      JMP.IF[EQL] TO [T5.7]         ;GO TO NEXT SECTION
U 1955, 8995,99 :7890      ASHL WR[0]           ;INCREMENT COUNTER
U 1956, 23D0,15 :7891      MOV WR[0] TO LS[T11]       ;RESTORE COUNTER
U 1957, 3E16,15 :7892      JMP [T5.6]             ;DO NEXT BIT
U 1958, 0993,F4 :7893
:7894
T5.7:     JSR [INC.EN]           ;INCREMENT ERROR NUMBER TO 7
U 1959, 8870,5C :7895
:7896
LOOP.T5.7: MOV LS[ONES] TO WR[1] ;SET ALL BITS IN WR1
U 195A, 369E,95 :7897      MOV WR[1] TO LS[T13]       ;SET ALL BITS IN LS LOCATION T13
U 195B, BE1A,95 :7898      CLR WR[0]           ;CLEAR ALL BITS IN WRO
U 195C, 2F80,15 :7899      JSR [ISSUE.SS]         ;ISSUE SCOPE SYNC
U 195D, 8870,9C :7900      BIS LS[T13] TO WR[0]       ;TRY BIS INS. FROM LOCATION 13 IN LS
U 195E, C71A,15 :7901      XOR WR[0] WITH LS[ONES] TO Q, ;WERE ALL BITS SET?
:7902      DT(LONG)&SET.ALU.CC
U 195F, 4D9E,35 :7903      SKIP.IF[NEQ]           ;SKIP NEXT INSTRUCTION IF NOT
U 1960, DB00,01 :7904      JMP [T5.8]             ;GO TO NEXT SECTION
U 1961, 8996,74 :7905      JSR [MOV.EX.RE]         ;SET UP EXPECTED AND RECEIVED DATA
U 1962, 8870,0C :7906      MISC [SET.CP.ATTN]       ;REPORT ERROR 7 TO CONSOLE
U 1963, 10E0,15 :7907
:7908
WAIT.T5.7: JMP [WAIT.T5.7]       ;WAIT FOR CONSOLE TO RESPOND
U 1964, 8996,44 :7909      JMP [LOOP.T5.7]         ;LOOP ON ERROR IF ENABLED
U 1965, 0995,A4 :7910      JMP [NOLOOP.T5.7]        ;GO TO NEXT SECTION IF NOT
U 1966, 0996,A4 :7910
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 5 - BIS Instru M 14
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 5 - BIS Instruction (M8390 CPU Module)

Fiche 1 Frame M14

Sequence 181
Rev 01.00

Page 175

```
U 1967, 36A0,15 :7911 T5.8: MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:7912 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
:7913 DT(LONG)&SET.ALU.CC
U 1968, CD82,35 :7914 JMP.IF[EQL] TO [LOOP.T5.7] ;LOOP ON ERROR IF SO
U 1969, 8995,A9 :7915 NOLOOP.T5.7: JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 8
:7916 LOOP.T5.8: MOV LS[L30] TO WR[0] ;GET DATA FROM TEST LOCATION
:7917 MOV WR[0] TO LS[T8] ;AND SAVE IT
:7918 MOV LS[ONES] TO WR[1] ;SET ALL BITS IN WR1
:7919 MOV WR[1] TO LS[L30] ;SET ALL BITS IN LS LOCATION 30
U 196B, 3660,15 :7920 CLR WR[0] ;CLEAR ALL BITS IN WRO
U 196C, 3E10,15 :7921 JSR [ISSUE.SS] ;ISSUE SCOPE SYNC
U 196D, 369E,95 :7922 BIS LS[L30] TO WR[0] ;TRY BIS INS. FROM LS LOCATION 30
U 196E, 3E60,95 :7923 XOR WR[0] WITH LS[ONES] TO Q, ;WERE ALL BITS SET?
U 196F, 2F80,15 :7924 DT(LONG)&SET.ALU.CC
U 1970, 8870,9C :7925 SKIP.IF[NEQ] ;SKIP NEXT INSTRUCTION IF NOT
U 1971, 4760,15 :7926 JMP [T5.8A]
:7927 JSR [MOV.EX.RE] ;SET UP EXPECTED AND RECEIVED DATA
:7928 MISC [SET.CP.ATTN] ;REPORT ERROR 8 TO CONSOLE
U 1972, 4D9E,35 :7929 WAIT.T5.8: JMP [WAIT.T5.8] ;WAIT FOR CONSOLE TO RESPOND
U 1973, DB00,01 :7930 JMP [LOOP.T5.8] ;LOOP ON ERROR IF ENABLED
U 1974, 8997,A4 :7931 JMP [T5.8B] ;GO TO END OF TEST IF NOT
U 1975, 8870,0C :7932 T5.8A: MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:7933 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
:7934 DT(LONG)&SET.ALU.CC
U 1976, 10E0,15 :7935 JMP.IF[EQL] TO [LOOP.T5.7] ;LOOP ON ERROR IF SO
:7936 T5.8B: MOV LS[T8] TO WR[0] ;GET OLD DATA
:7937 MOV WR[0] TO LS[L30] ;AND RESTORE IT
U 197A, 36A0,15 :7938 END.T5:
:7939
:7940
:7941
:7942
:7943
:7944
```


Page "TEST 6 - BIC Instruction (M8390 CPU Module)"

++

Test Description:

This test checks the BIC instruction that is used by the error routine. The BIC instruction used is BIC LS[] to WR[] and the 2901 is set up for the following mode: SRC=D,A DST=WRITE.B.F FUNC=R NOT AND S CIN=CIN. The data path is as follows: The contents of a WR is "ANDed" with the complement of the contents of the LS address specified by the u-instruction. The WR used is also specified in the u-instruction. The result is written back into the same WR and the result is also put on the Y BUS. The result is compared with a location in LS using the previously tested XOR WR WITH LS TO Q instruction.

Assumptions:

It is assumed that the XOR WR WITH LS TO Q instruction is working correctly because it was tested in a previous test. It is also assumed that the LS is previously loaded correctly with constants (the LS is loaded when the overlay is loaded and also at the end of the LS test).

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Try BIC instruction with data of 0 from LS and 0 in the WR. Check result for 0 with skip on ALU not equal 0 in SCTL field.
- 3) Check BIC with 0 from LS and all ones in the WR. Check result for non-zero with skip on ALU=0 in SCTL field.
- 4) Using XOR WR WITH LS TO Q, check result for all ones.
- 5) Check BIC with all ones from LS and 0 in the WR. Check result for all zeros.
- 6) Check BIC with all ones from LS and all ones in the WR. Check result for all zeros again.
- 7) Using a shifting 1 pattern from LS, check that each bit can be cleared separately. Repeat until all 32 bits have been tested. This check is done using index mode and the OS reg.
- 8) Check BIC with data of all ones from LS location in range 0-1F and all ones in the WR.
- 9) Check BIC with data of all ones from LS location in range 40-5F and all ones in the WR.

Errors:

- Error 1 - BIC LS TO WR failed with data of 0 and 0.
Error 2 - BIC LS TO WR failed with data of 0 and all ones. SCTL=skip on ALU=0 skipped.
Error 3 - BIC LS TO WR failed with data of 0 and all ones. Failed to store all ones in the WR.
Error 4 - BIC LS TO WR failed with data of all ones and 0.
Error 5 - BIC LS TO WR failed with data of all ones and all ones.
Error 6 - BIC LS TO WR failed with data of shifting ones and all ones.

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 6 - BIC Instru 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 6 - BIC Instruction (M8390 CPU Module)

Fiche 1 Frame B15

Sequence 183 Page 177
Rev 01.00

:8000
:8001
:8002
:8003
:8004
:8005
:8006
:8007
:8008
:8009
:8010
:8011
:8012
:8013
:8014
:8015
:8016
:8017
:8018
:8019
:8020
:8021
:8022
:8023
:8024
:8025
:8026
:8027
:8028
:8029
:8030
:8031
:8032
:8033
:8034
:8035
:8036
:8037
:8038
:8039
:8040
:8041
:8042
:8043
:8044
:8045
:8046
:8047
:8048
:8049
:8050
:8051
:8052
:8053
:8054

Error 7 - BIC LS TO WR failed with data from LS location in range 0-1F
Error 8 - BIC LS TO WR failed with data from LS location in range 40-5F

Debug:

Error 1 - This indicates a basic failure of the 2901 or it's control lines. The most likely problem is with the control lines or drivers. Referring to the chart at the beginning of this listing and this test description, check all control lines going to the 2901. If there is a problem here, check the output directly out of the DEST CTL ROM and the SRC.ALU CTL PAL. The inputs to these ICs, from CSR 22 through CSR 14 should be 1 0001 0101(B) respectively. This check should be made with the CPU single stepped to the BIC instruction. If the control lines are OK then suspect the 2901 itself. The error in this case will most likely be with one bit or a group of 4 bits.

Error 2 - If this is the first failure, the 2901 itself is the most likely problem. However if all bits are failing, the control signals should be checked out as described in error 1 above.

Error 3 - Same as error 2.

Error 4 - Same as error 2.

Error 5 - Same as error 2.

Error 6 - This error indicates either a problem with the 2901 or the DEST CTL ROM or SRC.ALU CTL PAL. The input to these IC's differs by one bit from the previous BIC instructions since the shifting one pattern uses index mode into the LS. CSR 22 through CSR 14 going to the ROM and PAL is 1 0001 0111(B) respectively. Since a new location in the ROM is being addressed and a new pattern is going to the PAL, these areas are suspect. If all bits are not failing, suspect an internal fault in the respective 2901.

Error 7 - Same as error 6 except that CSR 22 through 14 going to the ROM and PAL is 1 0001 0100(B).

Error 8 - Same as error 6 except that CSR 22 through 14 going to the ROM and PAL is 1 0001 0110(B).

--
T.6:

U 197F, B65E,15
U 1980, 3E8D,15
U 1981, 10E0,15
U 1982, 0998,24
U 1983, 2F80,15
U 1984, 3E8A,15
U 1985, BEA0,15
U 1986, 2040,15
U 1987, BE82,15
U 1988, 2040,15
U 1989, 3E8C,15

WAIT.T6.0:

MOV LS[BEGIN.TEST] TO WR[0] :SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] :SET BIT15 IN CNTRL AND STATUS WORD
MISC [SET.CP.ATTN] :BIT15 INDICATES START OF TEST TO CONSOLE
:ISSUE CPU ATTN TO CONSOLE
JMP [WAIT.T6.0] :LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR WR[0] :SET WRO TO 0
MOV WR[0] TO LS[ERROR.MASK] :CLEAR ERROR MASK.
MOV WR[0] TO LS[PREVIOUS.ERROR] :CLEAR PREVIOUS ERROR NUMBER
INC WR[0] :SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER] :SET ERROR NUMBER TO FIRST ERROR
INC WR[0] :SET WRO TO 2
MOV WR[0] TO LS[MODULE.NUM] :SET UP MODULE CODE FOR CPU MODULE

U 198A, B63E,15	:8055	MOV LS[ERR.CON] TO WR[0]	:LOAD WRO WITH A CONSTANT THAT SETS BITS 8,10, & 23
U 198B, 3E80,15	:8056	MOV WR[0] TO LS[CONTROL.STATUS]	:LOAD CONTROL AND STATUS WORD. BIT 8
	:8057		:SET INDICATES AN ERROR, BIT 10 SET
	:8058		:INDICATES CONSOLE CONTROLS ERROR
	:8059		:LOOPING.
	:8060		:THIS WORD IS LOADED NOW SO THAT IT NEEDN'T
	:8061		:BE LOADED AGAIN IF AN ERROR OCCURS.
	:8062		
U 198C, 2F80,15	:8063	LOOP.T6.1:	
U 198D, 8870,9C	:8064	CLR WR[0]	:CLEAR OUT WRO
	:8065	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC
U 198E, 459C,35	:8066	BIC LS[ZERO] TO WR[0],	:TRY BIC INS. WITH DATA OF ZERO
U 198F, DB00,01	:8067	DT(LONG)&SET.ALU.CC	
U 1990, 8999,74	:8068	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF WRO IS NOT ZERO
U 1991, 2F82,95	:8069	JMP [T6.2]	:GO TO NEXT SECTION
U 1992, 8870,0C	:8070	CLR WR[1]	:PUT EXPECTED DATA IN WR1
U 1993, 10E0,15	:8071	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
	:8072	MISC [SET.CP.ATTN]	:REPORT ERROR1 TO CONSOLE
U 1994, 8999,44	:8073	WAIT.T6.1:	
U 1995, 8998,C4	:8074	JMP [WAIT.T6.1]	:WAIT FOR CONSOLE TO RESPOND
U 1996, 0999,A4	:8075	JMP [LOOP.T6.1]	:LOOP ON ERROR IF ENABLED
	:8076	JMP [NOLOOP.T6.1]	:GOTO NEXT SECTION IF NOT
U 1997, 36A0,15	:8077	T6.2:	
	:8078	MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR NUMBER
U 1998, CD82,35	:8079	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1999, 0998,C9	:8080	DT(LONG)&SET.ALU.CC	
	:8081	JMP.IF[EQL] TO [LOOP.T6.1]	:LOOP ON ERROR IF SO
U 199A, 8870,5C	:8082	NOLOOP.T6.1:	
	:8083	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 2
U 199B, B69E,15	:8084	LOOP.T6.2:	
U 199C, 8870,9C	:8085	MOV LS[ONES] TO WR[0]	:SET DATA OF ALL ONES IN WRO
	:8086	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC
U 199D, 459C,35	:8087	BIC LS[ZERO] TO WR[0],	:TRY BIC INS WITH DATA OF ZERO
U 199E, 5B00,09	:8088	DT(LONG)&SET.ALU.CC	
U 199F, 099A,64	:8089	SKIP.IF[EQL]	:SKIP NEXT INSTRUCTION IF WRO WAS CLEARED
U 19A0, 369E,95	:8090	JMP [T6.3]	:GO TO NEXT SECTION
U 19A1, 8870,0C	:8091	MOV LS[ONES] TO WR[1]	:PUT EXPECTED DATA IN WR1
U 19A2, 10E0,15	:8092	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
	:8093	MISC [SET.CP.ATTN]	:REPORT ERROR 2 TO CONSOLE
U 19A3, 099A,34	:8094	WAIT.T6.2:	
U 19A4, 8999,B4	:8095	JMP [WAIT.T6.2]	:WAIT FOR CONSOLE TO RESPOND
U 19A5, 099A,94	:8096	JMP [LOOP.T6.2]	:LOOP ON ERROR IF ENABLED
	:8097	JMP [NOLOOP.T6.2]	:GOTO NEXT SECTION IF NOT
U 19A6, B6A0,95	:8098	T6.3:	
	:8099	MOV LS[PREVIOUS.ERROR] TO WR[1]	:GET PREVIOUS ERROR NUMBER
U 19A7, 4D82,B5	:8100	XOR WR[1] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 19A8, 0999,B9	:8101	DT(LONG)&SET.ALU.CC	
	:8102	JMP.IF[EQL] TO [LOOP.T6.2]	:LOOP ON ERROR IF SO
U 19A9, B642,95	:8103	NOLOOP.T6.2:	
U 19AA, 2042,95	:8104	MOV LS[#2] TO WR[1]	:SET WR1 TO 2
U 19AB, 3E82,95	:8105	INC WR[1]	:SET WR1 TO 3
	:8106	MOV WR[1] TO LS[ERROR.NUMBER]	:SET ERROR NUMBER TO 3
U 19AC, 4D9E,35	:8107	XOR WR[0] WITH LS[ONES] TO Q,	:IS WRO STILL ALL ONES?
U 19AD, DB00,01	:8108	DT(LONG)&SET.ALU.CC	
U 19AE, 899B,54	:8109	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF NOT
		JMP [T6.4]	:GO TO NEXT SECTION

U 19AF, 369E,95	:8110	MOV LS[ONES] TO WR[1]	:MOVE EXPECTED DATA TO WR1
U 19B0, 8870,0C	:8111	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 19B1, 10E0,15	:8112	MISC [SET.CP.ATTN]	:REPORT ERROR 3 TO CONSOLE
	:8113	WAIT.T6.3:	
U 19B2, 099B,24	:8114	JMP [WAIT.T6.3]	:WAIT FOR CONSOLE TO RESPOND
U 19B3, 8999,84	:8115	JMP [LOOP.T6.2]	:LOOP ON ERROR IF ENABLED
U 19B4, 099B,84	:8116	JMP [NOLOOP.T6.3]	:GO TO NEXT SECTION IF NOT
	:8117	T6.4:	
U 19B5, 36A0,15	:8118	MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR NUMBER
	:8119	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 19B6, CD82,35	:8120	DT(LONG)&SET.ALU.CC	
U 19B7, 0999,B9	:8121	JMP.IF[EQL] TO [LOOP.T6.2]	:LOOP ON ERROR IF SO
	:8122	NOLOOP.T6.3:	
U 19B8, 8870,5C	:8123	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 4
	:8124	LOOP.T6.4:	
U 19B9, 2F80,15	:8125	CLR WR[0]	:CLEAR THE WORKING REGISTER
U 19BA, 8870,9C	:8126	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC
	:8127	BIC LS[ONES] TO WR[0],	:TRY BIC INS. WITH DATA OF ONES
U 19BB, C59E,35	:8128	DT(LONG)&SET.ALU.CC	
U 19BC, DB00,01	:8129	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF WRO NOT STILL CLEAR
U 19BD, 899C,44	:8130	JMP [T6.5]	:GO TO NEXT SECTION
U 19BE, 2F82,95	:8131	CLR WR[1]	:PUT EXPECTED DATA IN WR1
U 19BF, 8870,0C	:8132	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 19C0, 10E0,15	:8133	MISC [SET.CP.ATTN]	:REPORT ERROR 4 TO CONSOLE
	:8134	WAIT.T6.4:	
U 19C1, 899C,14	:8135	JMP [WAIT.T6.4]	:WAIT FOR CONSOLE TO RESPOND
U 19C2, 899B,94	:8136	JMP [LOOP.T6.4]	:LOOP ON ERROR IF ENABLED
U 19C3, 899C,74	:8137	JMP [NOLOOP.T6.4]	:GO TO NEXT SECTION IF NOT
	:8138	T6.5:	
U 19C4, 36A0,15	:8139	MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR NUMBER
	:8140	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 19C5, CD82,35	:8141	DT(LONG)&SET.ALU.CC	
U 19C6, 099B,99	:8142	JMP.IF[EQL] TO [LOOP.T6.4]	:LOOP ON ERROR IF SO
	:8143	NOLOOP.T6.4:	
U 19C7, 8870,5C	:8144	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 5
	:8145	LOOP.T6.5:	
U 19C8, B69E,15	:8146	MOV LS[ONES] TO WR[0]	:SET ALL BITS IN WRO
U 19C9, 8870,9C	:8147	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC
	:8148	BIC LS[ONES] TO WR[0],	:TRY BIC INS. WITH DATA OF ONES
U 19CA, C59E,35	:8149	DT(LONG)&SET.ALU.CC	
U 19CB, DB00,01	:8150	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF WRO NOT CLEAR
U 19CC, 899D,34	:8151	JMP [T6.6.0]	:GO TO NEXT SECTION
U 19CD, 2F82,95	:8152	CLR WR[1]	:PUT EXPECTED DATA IN WR1
U 19CE, 8870,0C	:8153	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 19CF, 10E0,15	:8154	MISC [SET.CP.ATTN]	:REPORT ERROR 5 TO CONSOLE
	:8155	WAIT.T6.5:	
U 19D0, 899D,04	:8156	JMP [WAIT.T6.5]	:WAIT FOR CONSOLE TO RESPOND
U 19D1, 899C,84	:8157	JMP [LOOP.T6.5]	:LOOP ON ERROR IF ENABLED
U 19D2, 899D,64	:8158	JMP [NOLOOP.T6.5]	:GO TO NEXT SECTION IF NOT
	:8159	T6.6.0:	
U 19D3, 36A0,15	:8160	MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR NUMBER
	:8161	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 19D4, CD82,35	:8162	DT(LONG)&SET.ALU.CC	
U 19D5, 099C,89	:8163	JMP.IF[EQL] TO [LOOP.T6.5]	:LOOP ON ERROR IF SO
	:8164	NOLOOP.T6.5:	

U 19D6, 8870,5C :8165	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 6
U 19D7, 2F80,15 :8166	CLR WR[0]	:SET WRO TO 0
U 19D8, 3EF8,15 :8167	MOV WR[0] TO LS[OS]	:CLEAR THE OS REGISTER FOR INDEXING
U 19D9, B640,15 :8168	MOV LS[11] TO WR[0]	
U 19DA, 3E16,15 :8169	MOV WR[0] TO LS[T11]	:RESET COUNTER TO ONE
	T6.6:	
U 19DB, 2F80,15 :8170	CLR WR[0]	
U 19DC, BEA0,15 :8171	MOV WR[0] TO LS[PREVIOUS.ERROR]	:CLEAR PREVIOUS ERROR
	LOOP.T6.6:	
U 19DD, 2F80,15 :8172	CLR WR[0]	:CLEAR OUT ALL SET BITS IN WRO
U 19DE, 36F6,15 :8173	MOV LS[SHIFT.OS(4-0)] TO WR[0]	:SET BIT TO BE CLEARED IN WRO
U 19DF, 8870,9C :8174	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC
	BIC LS[SHIFT.OS(4-0)] TO WR[0],	:TRY BIC USING SINGLE BIT DATA FROM OS
U 19E0, 45F6,35 :8175	DT(LONG)&SET.ALU.CC	
U 19E1, DB00,01 :8176	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF BIT WAS NOT CLEARED
U 19E2, 899E,94 :8177	JMP [T6.6A]	:GO TO SET UP FOR NEXT BIT
U 19E3, 2F82,95 :8178	CLR WR[1]	:SET UP EXPECTED DATA IN WR1
U 19E4, 8870,0C :8179	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 19E5, 10E0,15 :8180	MISC [SET.CP.ATTN]	:REPORT ERROR 6 TO CONSOLE
	WAIT.T6.6:	
U 19E6, 899E,64 :8181	JMP [WAIT.T6.6]	:WAIT FOR CONSOLE TO RESPOND
U 19E7, 099D,D4 :8182	JMP [LOOP.T6.6]	:LOOP ON ERROR IF ENABLED
U 19E8, 899E,C4 :8183	JMP [NLOOP.T6.6]	
	T6.6A:	
U 19E9, 36A0,15 :8184	MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR NUMBER
	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 19EA, CD82,35 :8185	DT(LONG)&SET.ALU.CC	
U 19EB, 899D,D9 :8186	JMP.IF[EQL] TO [LOOP.T6.6]	:LOOP ON ERROR IF SO
	NLOOP.T6.6:	
U 19EC, 3616,95 :8187	MOV LS[T11] TO WR[1]	:GET INDEX
	XOR WR[1] WITH LS[T10] TO Q,	:IS IT EQUAL TO 32?
U 19ED, 4D14,B5 :8188	DT(LONG)&SET.ALU.CC	
U 19EE, 899F,59 :8189	JMP.IF[EQL] TO [T6.7]	:GO TO END OF TEST IF SO
U 19EF, A3D0,95 :8190	ASHL WR[1]	:INCREMENT COUNTER
U 19F0, BE16,95 :8191	MOV WR[1] TO LS[T11]	:RESTORE COUNTER
U 19F1, 36F8,95 :8192	MOV LS[OS] TO WR[1]	
U 19F2, 2042,95 :8193	INC WR[1]	:INCREMENT INDEX
U 19F3, BEF8,95 :8194	MOV WR[1] TO LS[OS]	:FOR NEXT BIT
U 19F4, 099D,B4 :8195	JMP [T6.6]	:DO AGAIN
	T6.7:	
U 19F5, 8870,5C :8196	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 7
	LOOP.T6.7:	
U 19F6, B69E,15 :8197	MOV LS[ONES] TO WR[0]	:SET ALL BITS IN WRO
U 19F7, 3E10,15 :8198	MOV WR[0] TO LS[T8]	:SET ALL BITS IN LS LOCATION 8
	BIC LS[T8] TO WR[0],	:TRY BIC INS. FROM LS LOCATION 8
U 19F8, C510,35 :8199	DT(LONG)&SET.ALU.CC	
U 19F9, DB00,01 :8200	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF WRO WAS NOT CLEARED
U 19FA, 89A0,14 :8201	JMP [T6.8]	
U 19FB, 369E,95 :8202	MOV LS[ONES] TO WR[1]	:PUT EXPECTED DATA IN WR1
U 19FC, 8870,0C :8203	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 19FD, 10E0,15 :8204	MISC [SET.CP.ATTN]	:REPORT ERROR 7 TO CONSOLE
	WAIT.T6.7:	
U 19FE, 899F,E4 :8205	JMP [WAIT.T6.7]	:WAIT FOR CONSOLE TO RESPOND
U 19FF, 099F,64 :8206	JMP [LOOP.T6.7]	:LOOP ON ERROR IF ENABLED
U 1A00, 89A0,44 :8207	JMP [NLOOP.T6.7]	:GO TO NEXT SECTION IF NOT

```

:8220 T6.8:
U 1A01, 36A0,15 :8221 MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:8222 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1A02, CD82,35 :8223 DT(LONG)&SET.ALU.CC
U 1A03, 899F,69 :8224 JMP.IF[EQ] TO [LOOP.T6.7] ;LOOP ON ERROR IF SO
:8225 NOLOOP.T6.7:
U 1A04, 8870,5C :8226 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 8
:8227 LOOP.T6.8:
U 1A05, B660,95 :8228 MOV LS[L30] TO WR[1] ;GET DATA FROM TEST LOCATION
U 1A06, BE10,95 :8229 MOV WR[1] TO LS[T8] ;AND SAVE IT
U 1A07, B69E,15 :8230 MOV LS[ONES] TO WR[0] ;SET ALL BITS IN WRO
U 1A08, BE60,15 :8231 MOV WR[0] TO LS[L30] ;SET ALL BITS IN LS LOCATION 30
:8232 BIC LS[BIT16] TO WR[0], ;TRY BIC INS. FROM LS LOCATION 30
:8233 DT(LONG)&SET.ALU.CC
U 1A09, 4560,35 :8234 SKIP.IF[NEQ] ;SKIP NEXT INSTRUCTION IF WRO WAS NOT CLEARED
U 1A0A, DB00,01 :8235 JMP [T6.8A]
U 1A0B, 09A1,24 :8236 MOV LS[ONES] TO WR[1] ;PUT EXPECTED DATA IN WR1
U 1A0C, 369E,95 :8237 JSR [MOV.EX.RE] ;SET UP EXPECTED AND RECEIVED DATA
U 1A0D, 8870,0C :8238 MISC [SET.CP.ATTN] ;REPORT ERROR 8 TO CONSOLE
:8239 WAIT.T6.8:
U 1A0F, 09A0,F4 :8240 JMP [WAIT.T6.8] ;WAIT FOR CONSOLE TO RESPOND
U 1A10, 09A0,54 :8241 JMP [LOOP.T6.8] ;LOOP ON ERROR IF ENABLED
U 1A11, 89A1,54 :8242 JMP [T6.8B] ;GOTO END OF TEST IF NOT
:8243 T6.8A:
U 1A12, 36A0,15 :8244 MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:8245 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1A13, CD82,35 :8246 DT(LONG)&SET.ALU.CC
U 1A14, 899F,69 :8247 JMP.IF[EQ] TO [LOOP.T6.7] ;LOOP ON ERROR IF SO
:8248 T6.8B:
U 1A15, 3610,95 :8249 MOV LS[T8] TO WR[1] ;GET DATA FROM TEST LOCATION
U 1A16, 3E60,95 :8250 MOV WR[1] TO LS[L30] ;AND RESTORE IT
:8251 END.T6:
  
```


:8252 Page "TEST 7 - BIT Instruction (M8390 CPU Module)"

:8253 ++

:8254
:8255 Test Description:

:8256
:8257 This test checks the BIT instruction that is used by the error routine.
:8258 The BIT instruction used is BIT WR[] WITH LS[] and the 2901 is set up
:8259 for the following mode: SRC=D, A DST=NOP FUNC=R AND S CIN=CIN.
:8260 The data path is as follows: The contents of a WR is "ANDed" with
:8261 the contents of the LS address specified by the u-instruction. The
:8262 WR used is also specified in the u-instruction. The result is put on
:8263 the Y BUS. Since the result is not stored anywhere, only the
:8264 condition codes can be checked in this test (only the Z bit is check-
:8265 ed). However the WR is checked to assure that it was not modified.
:8266

:8267 Assumptions:

:8268
:8269 It is assumed that the LS is previously loaded correctly with constants
:8270 (the LS is loaded when the overlay is loaded and also at the end of the
:8271 LS test).
:8272

:8273 Test Steps:

- :8274
:8275 1) Set up the error mask, error number, and module number in LS (for
:8276 error printouts) and clear previous error number in LS.
:8277 2) Try BIT instruction with data of 0 from LS and 0 in the WR. Check
:8278 result for 0 with skip on ALU not equal 0 in SCTL field.
:8279 3) Check BIT with 0 from LS and all ones in the WR. Check result for
:8280 0 with skip on ALU not equal 0 in SCTL field.
:8281 4) Check that the WR specified is still ones (has not been modified).
:8282 5) Check BIT with all ones from LS and 0 in the WR. Check result for
:8283 0 with skip on ALU not equal 0 in SCTL field.
:8284 6) Check BIT with all ones from LS and all ones in the WR. Check
:8285 result for non-zero with skip on ALU=0 in SCTL field.
:8286 7) Using a shifting 1 pattern from LS, and the same shifting pattern
:8287 in the WR check that each bit can clear the Z bit individually.
:8288 Repeat until all 32 bits have been tested. This check is done
:8289 using index mode and the OS reg.
:8290

:8291 Errors:

- :8292
:8293 Error 1 - BIT WR WITH LS failed with data of 0 and 0.
:8294 Error 2 - BIT WR WITH LS failed with data of 0 and all ones.
:8295 Error 3 - BIT WR WITH LS modified the WR.
:8296 Error 4 - BIT WR WITH LS failed with data of all ones and 0.
:8297 Error 5 - BIT WR WITH LS failed with data of all ones and all ones.
:8298 Error 6 - BIT WR WITH LS failed with data of shifting ones and all
:8299 ones.
:8300 Error 7 - BIT WR WITH LS failed with data in LS range 0-1F
:8301 Error 8 - BIT WR WITH LS failed with data in LS range 50-6F
:8302

:8303 Debug:

- :8304
:8305 Error 1 - This indicates a basic failure of the 2901 or it's control
:8306 lines. The most likely problem is with the control lines or drivers.

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 7 - BIT Instru 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 7 - BIT Instruction (M8390 CPU Module)

Fiche 1 Frame H15

Sequence 189
Rev 01.00 Page 183

:8307 : Referring to the chart at the beginning of this listing and this test
:8308 : description, check all control lines going to the 2901. If there is
:8309 : a problem here, check the output directly out of the DEST CTL ROM and
:8310 : the SRC.ALU CTL PAL. The inputs to these ICs, from CSR 22 through CSR
:8311 : 14 should be 1 0110 0101(B) respectively. This check should be made
:8312 : with the CPU single stepped to the BIT instruction. If the control
:8313 : lines are OK then suspect the 2901 itself. The error in this case will
:8314 : most likely be with one bit or a group of 4 bits.

:8315 :
:8316 : Error 2 - If this is the first failure, the 2901 itself is the most
:8317 : likely problem. However if all bits are failing, the control signals
:8318 : should be checked out as described in error 1 above.

:8319 :
:8320 : Error 3 - This error indicates that the WR specified was written into
:8321 : even though the DST was a NOP. First check the DST CTL lines at the
:8322 : 2901. Follow these back to the DEST CTL ROM or the SRC.ALU CTL PAL
:8323 : if they are in error. See error 1. If the control lines are OK, then
:8324 : suspect a bad 2901.

:8325 :
:8326 : Error 4 - Same as error 2.

:8327 :
:8328 : Error 5 - Same as error 2.

:8329 :
:8330 : Error 6 - This error indicates either a problem with the 2901 or the
:8331 : DEST CTL ROM or SRC.ALU CTL PAL. The input to these IC's differs by
:8332 : one bit from the previous BIT instructions since the shifting one
:8333 : pattern uses index mode into the LS. CSR 22 through CSR 14 going to
:8334 : the ROM and PAL is 1 0110 0111(B) respectively. Since a new location
:8335 : in the ROM is being addressed and a new pattern is going to the PAL,
:8336 : these areas are suspect. If all bits are not failing, suspect an
:8337 : internal fault in the respective 2901.

:8338 :
:8339 : Error 7 - Same as error 6 except that CSR 22 through 14 going to the
:8340 : ROM and PAL is 1 0110 0100(B).

:8341 :
:8342 : Error 8 - Same as error 6 except that CSR 22 through 14 going to the
:8343 : ROM and PAL is 1 0110 0110(B).

:8344 :
:8345 :
:8346 : T.7:--

U 1A17, B65E,15 :8347

U 1A18, 3E80,15 :8348

U 1A19, 10E0,15 :8349

U 1A1A, 89A1,A4 :8350

U 1A1B, 2F80,15 :8351

U 1A1C, 3E8A,15 :8352

U 1A1D, BEA0,15 :8353

U 1A1E, 2040,15 :8354

U 1A1F, BE82,15 :8355

U 1A20, 2040,15 :8356

U 1A21, 3E8C,15 :8357

U 1A22, 363C,15 :8358

U 1A23, 3E80,15 :8359

U 1A23, 3E80,15 :8360

T.7:--

WAIT.T7.0:

MOV LS[BEGIN.TEST] TO WR[0]
MOV WR[0] TO LS[CONTROL.STATUS]

MISC [SET.CP.ATTN]

JMP [WAIT.T7.0]

CLR WR[0]

MOV WR[0] TO LS[ERROR.MASK]

MOV WR[0] TO LS[PREVIOUS.ERROR]

INC WR[0]

MOV WR[0] TO LS[ERROR.NUMBER]

INC WR[0]

MOV WR[0] TO LS[MODULE.NUM]

MOV LS[ERR.CON.NA] TO WR[0]

MOV WR[0] TO LS[CONTROL.STATUS]

:SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
:SET BIT15 IN CNTROL AND STATUS WORD
:BIT15 INDICATES START OF TEST TO CONSOLE
:ISSUE CPU ATTN TO CONSOLE

:LOOP HERE WAITING FOR CONSOLE TO RESPOND

:SET WRO TO 0

:CLEAR ERROR MASK.

:CLEAR PREVIOUS ERROR NUMBER

:SET WRO TO 1

:SET ERROR NUMBER TO FIRST ERROR

:SET WRO TO 2

:SET UP MODULE CODE FOR CPU MODULE

:LOAD WRO WITH A CONSTANT THAT SETS BITS 8,10, & 23

:LOAD CONTROL AND STATUS WORD. BIT 8

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 7 - BIT Instru 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 7 - BIT Instruction (M8390 CPU Module)

Fiche 1 Frame I15

Sequence 190
Rev 01.00 Page 184

```
:8362  
:8363  
:8364  
:8365  
:8366  
:8367  
:8368  
U 1A24, 2F80,15 :8369  
U 1A25, 8870,9C :8370  
:8371  
U 1A26, D99C,35 :8372  
U 1A27, DB00,01 :8373  
U 1A28, 09A2,E4 :8374  
U 1A29, 09AA,8C :8375  
U 1A2A, 10E0,15 :8376  
:8377  
U 1A2B, 09A2,B4 :8378  
U 1A2C, 09A2,44 :8379  
U 1A2D, 89A3,14 :8380  
:8381  
U 1A2E, 36A0,15 :8382  
:8383  
U 1A2F, CD82,35 :8384  
U 1A30, 89A2,49 :8385  
:8386  
U 1A31, 8870,5C :8387  
:8388  
U 1A32, B69E,15 :8389  
U 1A33, 8870,9C :8390  
:8391  
U 1A34, D99C,35 :8392  
U 1A35, DB00,01 :8393  
U 1A36, 09A3,C4 :8394  
U 1A37, 09AA,8C :8395  
U 1A38, 10E0,15 :8396  
:8397  
U 1A39, 09A3,94 :8398  
U 1A3A, 89A3,24 :8399  
U 1A3B, 09A3,F4 :8400  
:8401  
U 1A3C, B6A0,95 :8402  
:8403  
U 1A3D, 4D82,B5 :8404  
U 1A3E, 09A3,29 :8405  
:8406  
U 1A3F, B642,95 :8407  
U 1A40, 2042,95 :8408  
U 1A41, 3E82,95 :8409  
:8410  
U 1A42, 4D9E,35 :8411  
U 1A43, DB00,01 :8412  
U 1A44, 89A4,A4 :8413  
U 1A45, 09AA,8C :8414  
U 1A46, 10E0,15 :8415  
:8416  
:  
: SET INDICATES AN ERROR, BIT 10 SET  
: INDICATES CONSOLE CONTROLS ERROR  
: LOOPING.  
: THIS WORD IS LOADED NOW SO THAT IT NEEDN'T  
: BE LOADED AGAIN IF AN ERROR OCCURS.  
  
LOOP.T7.1:  
CLR WR[0] :CLEAR OUT WRO  
JSR [ISSUE.SS] :SCOPE SYNC POINT  
BIT WR[0] WITH LS[ZERO], :TRY BIT INS. WITH DATA OF ZERO  
DT(LONG)&SET.ALU.CC  
SKIP.IF[NEQ] :SKIP NEXT INSTRUCTION IF Z BIT NOT SET  
JMP [T7.2] :GOTO NEXT SECTION  
JSR [T7.SET.PE] :MOVE ERROR NUMBER TO PREVIOUS ERROR  
MISC [SET.CP.ATTN] :REPORT ERROR 1 TO CONSOLE  
  
WAIT.T7.1:  
JMP [WAIT.T7.1] :WAIT FOR CONSOLE TO RESPOND  
JMP [LOOP.T7.1] :LOOP ON ERROR IF ENABLED  
JMP [NOLOOP.T7.1] :GO TO NEXT SECTION IF NOT  
  
T7.2:  
MOV LS[PREVIOUS.ERROR] TO WR[0] :GET PREVIOUS ERROR  
XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, :CURRENT ERROR?  
DT(LONG)&SET.ALU.CC  
JMP.IF[EQL] TO [LOOP.T7.1] :LOOP ON ERROR IF SO  
  
NOLOOP.T7.1:  
JSR [INC.EN] :INCREMENT ERROR NUMBER TO 2  
  
LOOP.T7.2:  
MOV LS[ONES] TO WR[0] :SET ALL BITS IN WRO  
JSR [ISSUE.SS] :ISSUE SCOPE SYNC SIGNAL  
BIT WR[0] WITH LS[ZERO], :TRY BIT INS. WITH DATA OF ZERO AND ALL BITS SET IN WRO  
DT(LONG)&SET.ALU.CC  
SKIP.IF[NEQ] :SKIP NEXT INSTRUCTION IF Z BIT NOT SET  
JMP [T7.3] :GO TO NEXT TEST  
JSR [T7.SET.PE] :MOVE ERROR NUMBER TO PREVIOUS ERROR  
MISC [SET.CP.ATTN] :REPORT ERROR 2 TO CONSOLE  
  
WAIT.T7.2:  
JMP [WAIT.T7.2] :WAIT FOR CONSOLE TO RESPOND  
JMP [LOOP.T7.2] :LOOP ON ERROR IF ENABLED  
JMP [NOLOOP.T7.2] :GO TO NEXT SECTION IF NOT  
  
T7.3:  
MOV LS[PREVIOUS.ERROR] TO WR[1] :GET PREVIOUS ERROR  
XOR WR[1] WITH LS[ERROR.NUMBER] TO Q, :CURRENT ERROR?  
DT(LONG)&SET.ALU.CC  
JMP.IF[EQL] TO [LOOP.T7.2] :LOOP ON ERROR IF SO  
  
NOLOOP.T7.2:  
MOV LS[#2] TO WR[1] :SET WR1 TO 2  
INC WR[1] :INCREMENT WRO TO 3  
MOV WR[1] TO LS[ERROR.NUMBER] :SET ERROR NUMBER TO 3  
XOR WR[0] WITH LS[ONES] TO Q, :ARE ALL BITS STILL SET IN WRO?  
DT(LONG)&SET.ALU.CC  
SKIP.IF[NEQ] :SKIP NEXT INSTRUCTION IF ANY BITS WERE CLEARED  
JMP [T7.4] :GOTO NEXT SECTION  
JSR [T7.SET.PE] :SET PREVIOUS ERROR TO ERROR NUMBER  
MISC [SET.CP.ATTN] :REPORT ERROR 3 TO CONSOLE  
  
WAIT.T7.3:
```


U 1A47, 09A4,74	:8417	JMP [WAIT.T7.3]	:WAIT FOR CONSOLE TO RESPOND
U 1A48, 89A3,24	:8418	JMP [LOOP.T7.2]	:LOOP ON ERROR IF ENABLED
U 1A49, 09A4,D4	:8419	JMP [NOLOOP.T7.3]	:GOTO NEXT SECTION IF NOT
	:8420		
U 1A4A, 36A0,15	:8421	T7.4: MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR
	:8422	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1A4B, CD82,35	:8423	DT(LONG)&SET.ALU.CC	
U 1A4C, 09A3,29	:8424	JMP.IF[EQL] TO [LOOP.T7.2]	:LOOP ON ERROR IF SO
	:8425	NOLOOP.T7.3:	
U 1A4D, 8870,5C	:8426	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 4
	:8427	LOOP.T7.4:	
U 1A4E, 2F80,15	:8428	CLR WR[0]	:CLEAR ALL BITS IN WRO
U 1A4F, 8870,9C	:8429	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC SIGNAL
	:8430	BIT WR[0] WITH LS[ONES],	:TRY BIT INS. WITH DATA OF ALL ONES
U 1A50, 599E,35	:8431	DT(LONG)&SET.ALU.CC	
U 1A51, DB00,01	:8432	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF Z BIT NOT CLEAR
U 1A52, 89A5,84	:8433	JMP [T7.5]	
U 1A53, 09AA,8C	:8434	JSR [T7.SET.PE]	:SET PREVIOUS ERROR TO ERROR NUMBER
U 1A54, 10E0,15	:8435	MISC [SET.CP.ATTN]	:REPORT ERROR 4 TO CONSOLE
	:8436	WAIT.T7.4:	
U 1A55, 09A5,54	:8437	JMP [WAIT.T7.4]	:WAIT FOR CONSOLE TO RESPOND
U 1A56, 09A4,E4	:8438	JMP [LOOP.T7.4]	:LOOP ON ERROR IF ENABLED
U 1A57, 89A5,B4	:8439	JMP [NOLOOP.T7.4]	:GO TO NEXT SECTION IF NOT
	:8440	T7.5:	
U 1A58, 36A0,15	:8441	MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR
	:8442	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1A59, CD82,35	:8443	DT(LONG)&SET.ALU.CC	
U 1A5A, 89A4,E9	:8444	JMP.IF[EQL] TO [LOOP.T7.4]	:LOOP ON ERROR IF SO
	:8445	NOLOOP.T7.4:	
U 1A5B, 8870,5C	:8446	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 5
	:8447	LOOP.T7.5:	
U 1A5C, BC9E,15	:8448	MOV LS[ONES] TO WR[0]	:SET ALL BITS IN WRO
U 1A5D, 8870,9C	:8449	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC SIGNAL
	:8450	BIT WR[0] WITH LS[ONES],	:TRY BIT INS. WITH DATA OF ONES AND ALL BITS SET IN WR
U 1A5E, 599E,35	:8451	DT(LONG)&SET.ALU.CC	
U 1A5F, 5B00,09	:8452	SKIP.IF[EQL]	:SKIP NEXT INSTRUCTION IF Z BIT SET
U 1A60, 09A6,64	:8453	JMP [T7.6.0]	:GOTO NEXT SECTION
U 1A61, 09AA,8C	:8454	JSR [T7.SET.PE]	:SET PREVIOUS ERROR TO ERROR NUMBER
U 1A62, 10E0,15	:8455	MISC [SET.CP.ATTN]	:REPORT ERROR 5 TO CONSOLE
	:8456	WAIT.T7.5:	
U 1A63, 09A6,34	:8457	JMP [WAIT.T7.5]	:WAIT FOR CONSOLE TO RESPOND
U 1A64, 09A5,C4	:8458	JMP [LOOP.T7.5]	:LOOP ON ERROR IF ENABLED
U 1A65, 09A6,94	:8459	JMP [NOLOOP.T7.5]	:GOTO NEXT SECTION IF NOT
	:8460	T7.6.0:	
U 1A66, 36A0,15	:8461	MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR
	:8462	XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1A67, CD82,35	:8463	DT(LONG)&SET.ALU.CC	
U 1A68, 89A5,C9	:8464	JMP.IF[EQL] TO [LOOP.T7.5]	:LOOP ON ERROR IF SO
	:8465	NOLOOP.T7.5:	
U 1A69, 8870,5C	:8466	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 6
U 1A6A, 2F80,15	:8467	CLR WR[0]	:SET WRO TO 0
U 1A6B, 3EF8,15	:8468	MOV WR[0] TO LS[OS]	:CLEAR OS REGISTER FOR INDEXING
	:8469	T7.6:	
U 1A6C, 2F80,15	:8470	CLR WR[0]	
U 1A6D, BEA0,15	:8471	MOV WR[0] TO LS[PREVIOUS.ERROR]	:CLEAR PREVIOUS ERROR


```

:8472 LOOP.T7.6:
U 1A6E, 2F80,15 :8473 CLR WR[0]
:8474 XOR WR[0] WITH LS[ZERO] TO Q, ;SET THE Z BIT
U 1A6F, CD9C,35 :8475 DT(LONG)&SET.ALU.CC
U 1A70, 36F6,15 :8476 MOV LS[SHIFT.OS(4-0)] TO WR[0] ;SET BIT IN WRO
U 1A71, 8870,9C :8477 JSR [ISSUE.SS] ;ISSUE SCOPE SYNC SIGNAL
:8478 BIT WR[0] WITH LS[SHIFT.OS(4-0)],
U 1A72, D9F6,35 :8479 DT(LONG)&SET.ALU.CC ;TRY BIT INS. WITH ONE BIT FROM LS AND SAME BIT IN WRO
U 1A73, 89A7,91 :8480 JMP.IF[NEQ] TO [T7.6A] ;GO TO NEXT SECTION IF Z BIT CLEAR
U 1A74, 09AA,8C :8481 JSR [T7.SET.PE] ;SET PREVIOUS ERROR TO ERROR NUMBER
U 1A75, 10E0,15 :8482 MISC [SET.CP.ATTN] ;REPORT ERROR 6 TO CONSOLE
:8483 WAIT.T7.6:
U 1A76, 89A7,64 :8484 JMP [WAIT.T7.6] ;WAIT FOR CONSOLE TO RESPOND
U 1A77, 89A6,E4 :8485 JMP [LOOP.T7.6] ;LOOP ON ERROR IF ENABLED
U 1A78, 89A7,C4 :8486 JMP [NOLOOP.T7.6] ;GO SET UP NEXT BIT IF NOT
:8487 T7.6A:
U 1A79, 36A0,15 :8488 MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR
:8489 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1A7A, CD82,35 :8490 DT(LONG)&SET.ALU.CC
U 1A7B, 09A6,E9 :8491 JMP.IF[EQL] TO [LOOP.T7.6] ;LOOP ON ERROR IF SO
:8492 NOLOOP.T7.6:
U 1A7C, 36F8,95 :8493 MOV LS[OS] TO WR[1] ;GET INDEX
U 1A7D, 2042,95 :8494 INC WR[1] ;INCREMENT INDEX
U 1A7E, C52C,95 :8495 BIC LS[FFFFFFF00] TO WR[1] ;CLEAR HIGH BITS
:8496 XOR WR[1] WITH LS[#20] TO Q, ;IS IT EQUAL TO 32.?
U 1A7F, CD4A,B5 :8497 DT(LONG)&SET.ALU.CC
U 1A80, 09A8,39 :8498 JMP.IF[EQL] TO [T7.7] ;GO TO END OF TEST IF SO
U 1A81, BEF8,95 :8499 MOV WR[1] TO LS[OS] ;PUT INDEX BACK IN OS REGISTER
U 1A82, 09A6,C4 :8500 JMP [T7.6] ;DO AGIAN WITH NEW INDEX
:8501 T7.7:
U 1A83, 8870,5C :8502 JSR [INC.EN] ;UPDATE ERROR NUMBER TO 7
:8503 LOOP.T7.7:
U 1A84, B69E,15 :8504 MOV LS[ONES] TO WR[0] ;SET ALL BITS IN WRO
U 1A85, 3E10,15 :8505 MOV WR[0] TO LS[T8] ;SET ALL BITS IN TEST LOCATION
U 1A86, 2F80,15 :8506 CLR WR[0] ;CLEAR OUT WRO
U 1A87, 8870,9C :8507 JSR [ISSUE.SS] ;ISSUE SCOPE SYNC SIGNAL
:8508 BIT WR[0] WITH LS[T8], ;TRY BIT INS WITH LS LOCATION IN RANGE 0-20
U 1A88, 5910,35 :8509 DT(LONG)&SET.ALU.CC
U 1A89, DB00,01 :8510 SKIP.IF[NEQ] ;SKIP NEXT INSTRUCTION IF Z BIT NOT CLEAR
U 1A8A, 09A9,04 :8511 JMP [T7.8] ;GO TO NEXT SECTION
U 1A8B, 09AA,8C :8512 JSR [T7.SET.PE] ;SET PREVIOUS ERROR
U 1A8C, 10E0,15 :8513 MISC [SET.CP.ATTN] ;REPORT ERROR 7 TO CONSOLE
:8514 WAIT.T7.7:
U 1A8D, 09A8,D4 :8515 JMP [WAIT.T7.7] ;WAIT FOR CONSOLE TO RESPOND
U 1A8E, 09A8,44 :8516 JMP [LOOP.T7.7] ;LOOP ON ERROR IF ENABLED
U 1A8F, 09A9,34 :8517 JMP [NOLOOP.T7.7] ;GO TO NEXT SECTION IF NOT
:8518 T7.8:
U 1A90, 36A0,15 :8519 MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR
:8520 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1A91, CD82,35 :8521 DT(LONG)&SET.ALU.CC
U 1A92, 89A8,49 :8522 JMP.IF[EQL] TO [LOOP.T7.7] ;LOOP ON ERROR IF SO
:8523 NOLOOP.T7.7:
U 1A93, 8870,5C :8524 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 8
:8525 LOOP.T7.8:
U 1A94, B660,95 :8526 MOV LS[L30] TO WR[1] ;GET DATA FROM TEST LOCATION
    
```

```

U 1A95, BE10.95 :8527      MOV WR[1] TO LS[T8]      ; AND SAVE IT
U 1A96, B69E.15 :8528      MOV LS[ONES] TO WR[0]      ; SET ALL BITS IN WRO
U 1A97, BE60.15 :8529      MOV WR[0] TO LS[L30]      ; SET ALL BITS IN TEST LOCATION
U 1A98, 2F80.15 :8530      CLR WR[0]                ; CLEAR OUT WRO
U 1A99, 8870.9C :8531      JSR [ISSUE.SS]             ; ISSUE SCOPE SYNC SIGNAL
:8532      BIT WR[0] WITH LS[L30],                     ; TRY BIT INS WITH LS LOCATION 50
:8533      DT(LONG)&SET.ALU.CC
U 1A9A, D960.35 :8534      SKIP IF[NEQ]                ; SKIP NEXT INSTRUCTION IF Z BIT NOT CLEAR
U 1A9B, DB00.01 :8535      JMP [NOLoop.T7.8]           ; GO TO NEXT SECTION
U 1A9C, 09AA.54 :8536      JSR [T7.SET.PE]             ; SET PREVIOUS ERROR
U 1A9D, 09AA.8C :8537      MISC [SET.CP.ATTN]          ; REPORT ERROR 8 TO CONSOLE
U 1A9E, 10E0.15 :8538      WAIT.T7.8:
:8539      JMP [WAIT.T7.8]                ; WAIT FOR CONSOLE TO RESPOND
U 1AA0, 89A9.44 :8540      JMP [LOOP.T7.8]             ; LOOP ON ERROR IF ENABLED
U 1AA1, 09AA.54 :8541      JMP [NOLoop.T7.8]           ; GO TO NEXT SECTION IF NOT
:8542      T7.8A:
U 1AA2, 36A0.15 :8543      MOV LS[PREVIOUS.ERROR] TO WR[0] ; GET PREVIOUS ERROR
:8544      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ; CURRENT ERROR?
:8545      DT(LONG)&SET.ALU.CC
U 1AA3, CD82.35 :8546      JMP IF[EQL] TO [LOOP.T7.8]    ; LOOP ON ERROR IF SO
U 1AA4, 09A9.49 :8547      NOLoop.T7.8:
:8548      MOV LS[T8] TO WR[1]            ; GET SAVED DATA
U 1AA5, 3610.95 :8549      MOV WR[1] TO LS[L30]        ; AND RESTORE IT
U 1AA6, 3E60.95 :8550      JMP [END.T7]               ; SKIP OVER SUBROUTINES
U 1AA7, 89AA.B4 :8551      T7.SET.PE:
:8552      MOV LS[ERROR.NUMBER] TO WR[0]    ; GET ERROR NUMBER
U 1AA8, 3682.15 :8553      MOV WR[0] TO LS[PREVIOUS.ERROR] ; PUT IT IN PREVIOUS ERROR
U 1AA9, BEA0.15 :8554      RETURN
U 1AAA, 5B00.14 :8555      END.T7:
  
```


:8556 .Page "TEST 8 - CLR LS Instruction (M8390 CPU Module)"
:8557 ++

:8558
:8559 : Test Description:

:8560
:8561 This test checks the CLR LS instruction. This is the last instruction
:8562 to be tested before the error subroutine can be used with confidence.
:8563 The 2901 is set up for the following mode: SRC=D,0 DST=NOP FUNC=R
:8564 AND S CIN=CIN. The data path is as follows: The data on the D BUS
:8565 is "ANDed" with 0 in the 2901 and output on the Y bus (outputs 0).
:8566 The LS has also been enabled via the LS CONTROL PAL due to this being
:8567 a BASIC instruction with an LS destination (DP field=20-3F). The
:8568 data from the Y BUS is written in LS thus clearing it.
:8569

:8570 : Assumptions:

:8571
:8572 It is assumed that the LS itself has been tested or is known to be
:8573 good.
:8574

:8575 : Test Steps:

- :8576
:8577 1) Set up the error mask, error number, and module number in LS (for
:8578 error printouts) and clear previous error number in LS.
:8579 2) Write all ones in a temp location of LS in the range 0-1F.
:8580 3) Execute a CLR LS and check the result.
:8581 4) Check that the WR specified (WRO) is still ones (has not been mod-
:8582 ified).
:8583 5) Repeat steps 2,3, & 4, for LS locations in the ranges 20-3F, 40-5F,
:8584 and 60-7F.
:8585

:8586 : Errors:

- :8587
:8588 Error 1 - CLR LS failed to clear a location in LS in the range 0-1F.
:8589 Error 2 - CLR LS modified a WR.
:8590 Error 3 - CLR LS failed to clear a location in LS in the range 20-3F.
:8591 Error 4 - CLR LS modified a WR.
:8592 Error 5 - CLR LS failed to clear a location in LS in the range 40-5F.
:8593 Error 6 - CLR LS modified a WR.
:8594 Error 7 - CLR LS failed to clear a location in LS in the range 60-7F.
:8595 Error 8 - CLR LS modified a WR.
:8596

:8597 : Debug:

:8598
:8599 Error 1 - The most likely cause of this failure is with the control
:8600 lines for the 2901. This instruction uses the same functions of the
:8601 2901 has been used before so the 2901 itself is not a likely
:8602 prospect (although anything is possible!). The control lines from
:8603 the DEST CTL ROM and the SRC.ALU CTL PAL may be in error. Refer to
:8604 the 2901 tables at the beginning of this listing for the correct
:8605 values of the control lines. The input to these IC's for CSR 22
:8606 through CSR 14 should be 1 1001 0100.
:8607

:8608 Error 2 - This error indicates that the WR specified was written into
:8609 even though the DST was a NOP. First check the DST CTL lines at the
:8610 2901. Follow these back to the DEST CTL ROM or the SRC.ALU CTL PAL


```

:8611 : if they are in error. See error 1. If the control lines are OK, then
:8612 : suspect a bad 2901.
:8613 :
:8614 : Error 3 - Same as error 1 except that the CSR 22 through 14 going to
:8615 : the PAL and ROM should be 1 1001 0101
:8616 :
:8617 : Error 4 - Same as error 2.
:8618 :
:8619 : Error 5 - Same as error 1 except that the CSR 22 through 14 going to
:8620 : the PAL and ROM should be 1 1001 0110
:8621 :
:8622 : Error 6 - Same as error 2.
:8623 :
:8624 : Error 7 - Same as error 1 except that the CSR 22 through 14 going to
:8625 : the PAL and ROM should be 1 1001 0111
:8626 :
:8627 : Error 8 - Same as error 2.
:8628 :
:8629 :
U 1AAB, B65E,15 :8630
U 1AAC, 3E80,15 :8631
:8632
U 1AAD, 10E0,15 :8633
:8634
U 1AAE, 89AA,E4 :8635
U 1AAF, 2F80,15 :8636
U 1AB0, 3E8A,15 :8637
U 1AB1, BEA0,15 :8638
U 1AB2, 2040,15 :8639
U 1AB3, BE82,15 :8640
U 1AB4, 2040,15 :8641
U 1AB5, 3E8C,15 :8642
U 1AB6, B63E,15 :8643
U 1AB7, 3E80,15 :8644
:8645
:8646
:8647
:8648
:8649
U 1AB8, 3660,15 :8650
U 1AB9, BE12,15 :8651
U 1ABA, B6E6,15 :8652
U 1ABB, BE14,15 :8653
:8654
U 1ABC, B640,15 :8655
U 1ABD, BE82,15 :8656
U 1ABE, B69E,15 :8657
U 1ABF, 3E10,15 :8658
U 1ACO, 8870,9C :8659
U 1AC1, E510,15 :8660
U 1AC2, 3E16,15 :8661
U 1AC3, B610,15 :8662
:8663
U 1AC4, 2000,35 :8664
U 1AC5, DB00,01 :8665

T.8:
MOV LS[BEGIN.TEST] TO WR[0] :SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] :SET BIT15 IN CNTROL AND STATUS WORD
:MISC [SET.CP.ATTN] :BIT15 INDICATES START OF TEST TO CONSOLE
:ISSUE CPU ATTN TO CONSOLE

WAIT.T8.0:
JMP [WAIT.T8.0] :LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR WR[0] :SET WRO TO 0
MOV WR[0] TO LS[ERROR.MASK] :CLEAR ERROR MASK.
MOV WR[0] TO LS[PREVIOUS.ERROR] :CLEAR PREVIOUS ERROR NUMBER
INC WR[0] :SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER] :SET ERROR NUMBER TO FIRST ERROR
INC WR[0] :SET WRO TO 2
MOV WR[0] TO LS[MODULE.NUM] :SET UP MODULE CODE FOR CPU MODULE
MOV LS[ERR.CON] TO WR[0] :LOAD WRO WITH A CONSTANT THAT SETS BITS 8,10, & 23
MOV WR[0] TO LS[CONTROL.STATUS] :LOAD CONTROL AND STATUS WORD. BIT 8
: SET INDICATES AN ERROR, BIT 10 SET
: INDICATES CONSOLE CONTROLS ERROR
: LOOPING.
: THIS WORD IS LOADED NOW SO THAT IT NEEDN'T
: BE LOADED AGAIN IF AN ERROR OCCURS.

MOV LS[L30] TO WR[0]
MOV WR[0] TO LS[T9] :SAVE LOCATION 30
MOV LS[L73] TO WR[0]
MOV WR[0] TO LS[T10] :SAVE LOCATION 73

LOOP.T8.1:
MOV LS[#1] TO WR[0]
MOV WR[0] TO LS[ERROR.NUMBER] :ERROR NUMBER = 1
MOV LS[ONES] TO WR[0] :SET ALL BITS IN WRO
MOV WR[0] TO LS[T8] :SET ALL BITS AT LS TEST LOCATION (RANGE 0-1F)
JSR [ISSUE.SS] :ISSUE SCOPE SYNC SIGNAL
CLR LS[T8] :TRY CLEAR INSTRUCTION
MOV WR[0] TO LS[T11] :SAVE WRO
MOV LS[T8] TO WR[0]
TST WR[0]
DT(LONG)&SET.ALU.CC
SKIP.IF[NEQ] :SKIP NEXT INSTRUCTION IF LOCATION WAS NOT CLEARED

```

U 1AC6, 89AC,E4 :8666	JMP [T8.2]	:GO TO NEXT SECTION
U 1AC7, 2F82,95 :8667	CLR WR[1]	:MOVE EXPECTED DATA TO WR1
U 1AC8, B610,15 :8668	MOV LS[T8] TO WR[0]	:MOVE RECEIVED DATA TO WRO
U 1AC9, 8870,0C :8669	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 1ACA, 10E0,15 :8670	MISC [SET.CP.ATTN]	:REPORT ERROR 1 TO CONSOLE
	WAIT.T8.1:	
U 1ACB, 89AC,B4 :8672	JMP [WAIT.T8.1]	:WAIT FOR CONSOLE TO RESPOND
U 1ACC, 89AB,C4 :8673	JMP [LOOP.T8.1]	:LOOP ON ERROR IF ENABLED
U 1ACD, 09AD,14 :8674	JMP [NOLOOP.T8.1]	:GO TO NEXT SECTION IF NOT
	T8.2:	
U 1ACE, B6A0,95 :8676	MOV LS[PREVIOUS.ERROR] TO WR[1]	:GET PREVIOUS ERROR NUMBER
	XOR WR[1] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1ACF, 4D82,B5 :8678	DT(LONG)&SET.ALU.CC	
U 1AD0, 09AB,C9 :8679	JMP.IF[EQL] TO [LOOP.T8.1]	:LOOP IF SO
	NOLOOP.T8.1:	
U 1AD1, B642,95 :8681	MOV LS[#2] TO WR[1]	
U 1AD2, 3E82,95 :8682	MOV WR[1] TO LS[ERROR.NUMBER]	:SET ERROR NUMBER TO 2
U 1AD3, B616,15 :8683	MOV LS[T11] TO WR[0]	:RESTORE WRO
	XOR WR[0] WITH LS[ONES] TO Q,	:COMPARE WRO WITH ALL ONES
	DT(LONG)&SET.ALU.CC	
U 1AD4, 4D9E,35 :8685	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF ALL BITS IN WRO WERE NOT STILL SET
U 1AD5, DB00,01 :8686	JMP [T8.3]	:GO TO NEXT SECTION
U 1AD6, 09AD,D4 :8687	MOV LS[ONES] TO WR[1]	:SET EXPECTED DATA IN WR1
U 1AD7, 369E,95 :8688	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 1AD8, 8870,0C :8689	MISC [SET.CP.ATTN]	:REPORT ERROR 2 TO CONSOLE
U 1AD9, 10E0,15 :8690		
	WAIT.T8.2:	
U 1ADA, 89AD,A4 :8692	JMP [WAIT.T8.2]	:WAIT FOR CONSOLE TO RESPOND
U 1ADB, 89AB,C4 :8693	JMP [LOOP.T8.1]	:LOOP ON ERROR IF ENABLED
U 1ADC, 89AE,04 :8694	JMP [NOLOOP.T8.2]	:GO TO NEXT SECTION IF NOT
	T8.3:	
U 1ADD, B6A0,95 :8696	MOV LS[PREVIOUS.ERROR] TO WR[1]	:GET PREVIOUS ERROR NUMBER
	XOR WR[1] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1ADE, 4D82,B5 :8698	DT(LONG)&SET.ALU.CC	
U 1ADF, 09AB,C9 :8699	JMP.IF[EQL] TO [LOOP.T8.1]	:LOOP IF SO
	NOLOOP.T8.2:	
U 1AE0, 8870,5C :8701	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 3
	LOOP.T8.3:	
U 1AE1, 3642,15 :8703	MOV LS[#2] TO WR[0]	
U 1AE2, 2040,15 :8704	INC WR[0]	
U 1AE3, BE82,15 :8705	MOV WR[0] TO LS[ERROR.NUMBER]	:ERROR NUMBER = 3
U 1AE4, B69E,15 :8706	MOV LS[ONES] TO WR[0]	:SET ALL BITS IN WRO
U 1AE5, 3E8A,15 :8707	MOV WR[0] TO LS[ERROR.MASK]	:SET ALL BITS AT LS TEST LOCATION (RANGE 20-3F)
U 1AE6, 8870,9C :8708	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC SIGNAL
U 1AE7, E58A,15 :8709	CLR LS[ERROR.MASK]	:TRY CLEAR INSTRUCTION
U 1AE8, 3E16,15 :8710	MOV WR[0] TO LS[T11]	:SAVE WRO
U 1AE9, B68A,15 :8711	MOV LS[ERROR.MASK] TO WR[0]	
	TST WR[0]	
U 1AEA, 2000,35 :8713	DT(LONG)&SET.ALU.CC	
U 1AEB, DB00,01 :8714	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF LOCATION WAS NOT CLEARED
U 1AEC, 89AF,44 :8715	JMP [T8.4]	:GO TO NEXT SECTION
U 1AED, 2F82,95 :8716	CLR WR[1]	:MOVE EXPECTED DATA TO WR1
U 1AEE, B68A,15 :8717	MOV LS[ERROR.MASK] TO WR[0]	:MOVE RECEIVED DATA TO WRO
U 1AEF, 8870,0C :8718	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 1AFO, 10E0,15 :8719	MISC [SET.CP.ATTN]	:REPORT ERROR 3 TO CONSOLE
	WAIT.T8.3:	
	:8720	

U 1AF1, 89AF,14	:8721	JMP [WAIT.T8.3]	:WAIT FOR CONSOLE TO RESPOND
U 1AF2, 09AE,14	:8722	JMP [LOOP.T8.3]	:LOOP ON ERROR IF ENABLED
U 1AF3, 89AF,74	:8723	JMP [NOLOOP.T8.3]	:GO TO NEXT SECTION IF NOT
	:8724		
U 1AF4, B6A0,95	:8725	T8.4: MOV LS[PREVIOUS.ERROR] TO WR[1]	:GET PREVIOUS ERROR NUMBER
	:8726	XOR WR[1] WITH LS[ERROR.NUMBER]	TO 0, :CURRENT ERROR?
U 1AF5, 4D82,B5	:8727	DT(LONG)&SET.ALU.CC	
U 1AF6, 89AE,19	:8728	JMP.IF[EQ] TO [LOOP.T8.3]	:LOOP IF SO
	:8729	NOLOOP.T8.3:	
U 1AF7, B644,95	:8730	MOV LS[#4] TO WR[1]	:SET WR1 TO 4
U 1AF8, 3E82,95	:8731	MOV WR[1] TO LS[ERROR.NUMBER]	:SET ERROR NUMBER TO 4
U 1AF9, B616,15	:8732	MOV LS[T11] TO WR[0]	:RESTORE WRO
	:8733	XOR WR[0] WITH LS[ONES] TO Q,	:COMPARE WRO WITH ALL ONES
U 1AFA, 4D9E,35	:8734	DT(LONG)&SET.ALU.CC	
U 1AFB, DB00,01	:8735	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF ALL BITS IN WRO WERE NOT STILL SET
U 1AFC, 89B0,34	:8736	JMP [T8.5]	:GO TO NEXT SECTION
U 1AFD, 369E,95	:8737	MOV LS[ONES] TO WR[1]	:SET EXPECTED DATA IN WR1
U 1AFE, 8870,0C	:8738	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 1AFF, 10E0,15	:8739	MISC [SET.CP.ATTN]	:REPORT ERROR 2 TO CONSOLE
	:8740	WAIT.T8.4:	
U 1B00, 89B0,04	:8741	JMP [WAIT.T8.4]	:WAIT FOR CONSOLE TO RESPOND
U 1B01, 09AE,14	:8742	JMP [LOOP.T8.3]	:LOOP ON ERROR IF ENABLED
U 1B02, 89B0,64	:8743	JMP [NOLOOP.T8.4]	:GO TO NEXT SECTION IF NOT
	:8744		
U 1B03, B6A0,95	:8745	T8.5: MOV LS[PREVIOUS.ERROR] TO WR[1]	:GET PREVIOUS ERROR NUMBER
	:8746	XOR WR[1] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1B04, 4D82,B5	:8747	DT(LONG)&SET.ALU.CC	
U 1B05, 89AE,19	:8748	JMP.IF[EQ] TO [LOOP.T8.3]	:LOOP IF SO
	:8749	NOLOOP.T8.4:	
U 1B06, 8870,5C	:8750	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 5
	:8751	LOOP.T8.5:	
U 1B07, 3644,15	:8752	MOV LS[#4] TO WR[0]	
U 1B08, 2040,15	:8753	INC WR[0]	
U 1B09, BE82,15	:8754	MOV WR[0] TO LS[ERROR.NUMBER]	:ERROR NUMBER = 5
U 1B0A, B69E,15	:8755	MOV LS[ONES] TO WR[0]	:SET ALL BITS IN WRO
U 1B0B, BE60,15	:8756	MOV WR[0] TO LS[L30]	:SET ALL BITS AT LS TEST LOCATION (RANGE 40-5F)
U 1B0C, 8870,9C	:8757	JSR [ISSUE.SS]	:ISSUE SCOPE SYNC SIGNAL
U 1B0D, 6560,15	:8758	CLR LS[L30]	:TRY CLEAR INSTRUCTION
U 1B0E, 3E16,15	:8759	MOV WR[0] TO LS[T11]	:SAVE WRO
U 1B0F, 3660,15	:8760	MOV LS[L30] TO WR[0]	
	:8761	TST WR[0],	
U 1B10, 2000,35	:8762	DT(LONG)&SET.ALU.CC	
U 1B11, DB00,01	:8763	SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF LOCATION WAS NOT CLEARED
U 1B12, 09B1,A4	:8764	JMP [T8.6]	:GO TO NEXT SECTION
U 1B13, 2F82,95	:8765	CLR WR[1]	:MOVE EXPECTED DATA TO WR1
U 1B14, 3660,15	:8766	MOV LS[L30] TO WR[0]	:MOVE RECEIVED DATA TO WRO
U 1B15, 8870,0C	:8767	JSR [MOV.EX.RE]	:SET UP EXPECTED AND RECEIVED DATA
U 1B16, 10E0,15	:8768	MISC [SET.CP.ATTN]	:REPORT ERROR 5 TO CONSOLE
	:8769	WAIT.T8.5:	
U 1B17, 89B1,74	:8770	JMP [WAIT.T8.5]	:WAIT FOR CONSOLE TO RESPOND
U 1B18, 09B0,74	:8771	JMP [LOOP.T8.5]	:LOOP ON ERROR IF ENABLED
U 1B19, 89B1,D4	:8772	JMP [NOLOOP.T8.5]	:GO TO NEXT SECTION IF NOT
	:8773		
U 1B1A, B6A0,95	:8774	T8.6: MOV LS[PREVIOUS.ERROR] TO WR[1]	:GET PREVIOUS ERROR NUMBER
	:8775	XOR WR[1] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?


```

U 1B1B, 4D82.B5 :8776      DT(LONG)&SET.ALU.CC
U 1B1C, 89B0.79 :8777      JMP.IF[EQL] TO [LOOP.T8.5]      ;LOOP IF SO
                                NOLOOP.T8.5:
U 1B1D, B644.95 :8778      MOV LS[#4] TO WR[1]      ;SET WR1 TO 4
U 1B1E, 2042.95 :8779      INC WR[1]      ;SET WR1 TO 5
U 1B1F, 2042.95 :8780      INC WR[1]      ;SET WR1 TO 6
U 1B20, 3E82.95 :8781      MOV WR[1] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO 6
U 1B21, B616.15 :8782      MOV LS[T11] TO WR[0]      ;RESTORE WRO
                                XOR WR[0] WITH LS[ONES] TO Q, ;COMPARE WRO WITH ALL ONES
U 1B22, 4D9E.35 :8783      DT(LONG)&SET.ALU.CC
U 1B23, DB00.01 :8784      SKIP.IF[NEQ]      ;SKIP NEXT INSTRUCTION IF ALL BITS IN WRO WERE NOT STILL SET
U 1B24, 89B2.B4 :8785      JMP [T8.7]      ;GO TO NEXT SECTION
U 1B25, 369E.95 :8786      MOV LS[ONES] TO WR[1]      ;SET EXPECTED DATA IN WR1
U 1B26, 8870.0C :8787      JSR [MOV.EX.RE]      ;SET UP EXPECTED AND RECEIVED DATA
U 1B27, 10E0.15 :8788      MISC [SET.CP.ATTN]      ;REPORT ERROR 6 TO CONSOLE
                                WAIT.T8.6:
U 1B28, 89B2.84 :8789      JMP [WAIT.T8.6]      ;WAIT FOR CONSOLE TO RESPOND
U 1B29, 09B0.74 :8790      JMP [LOOP.T8.5]      ;LOOP ON ERROR IF ENABLED
U 1B2A, 89B2.E4 :8791      JMP [NOLOOP.T8.6]      ;GO TO NEXT SECTION IF NOT
                                T8.7:
U 1B2B, B6A0.95 :8792      MOV LS[PREVIOUS.ERROR] TO WR[1] ;GET PREVIOUS ERROR NUMBER
                                XOR WR[1] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1B2C, 4D82.B5 :8793      DT(LONG)&SET.ALU.CC
U 1B2D, 89B0.79 :8794      JMP.IF[EQL] TO [LOOP.T8.5]      ;LOOP IF SO
                                NOLOOP.T8.6:
U 1B2E, 8870.5C :8795      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 7
                                LOOP.T8.7:
U 1B2F, B646.15 :8796      MOV LS[#8] TO WR[0]
U 1B30, 2100.15 :8797      DEC WR[0]
U 1B31, BE82.15 :8798      MOV WR[0] TO LS[ERROR.NUMBER] ;ERROR NUMBER = 7
U 1B32, B69E.15 :8799      MOV LS[ONES] TO WR[0]      ;SET ALL BITS IN WRO
U 1B33, 3EE6.15 :8800      MOV WR[0] TO LS[L73]      ;SET ALL BITS AT LS TEST LOCATION (RANGE 60-7F)
U 1B34, 8870.9C :8801      JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC SIGNAL
U 1B35, E5E6.15 :8802      CLR LS[L73]      ;TRY CLEAR INSTRUCTION
U 1B36, 3E16.15 :8803      MOV WR[0] TO LS[T11]      ;SAVE WRO
U 1B37, B6E6.15 :8804      MCV LS[L73] TO WR[0]
                                DT(LONG)&SET.ALU.CC
U 1B38, 2000.35 :8805      SKIP.IF[NEQ]      ;SKIP NEXT INSTRUCTION IF LOCATION WAS NOT CLEARED
U 1B39, DB00.01 :8806      JMP [T8.8]      ;GO TO NEXT SECTION
U 1B3A, 89B4.24 :8807      CLR WR[1]      ;MOVE EXPECTED DATA TO WR1
U 1B3B, 2F82.95 :8808      MOV LS[L73] TO WR[0]      ;MOVE RECEIVED DATA TO WRO
U 1B3C, B6E6.15 :8809      JSR [MOV.EX.RE]      ;SET UP EXPECTED AND RECEIVED DATA
U 1B3D, 8870.0C :8810      MISC [SET.CP.ATTN]      ;REPORT ERROR 7 TO CONSOLE
U 1B3E, 10E0.15 :8811      WAIT.T8.7:
                                JMP [WAIT.T8.7]      ;WAIT FOR CONSOLE TO RESPOND
U 1B3F, 89B3.F4 :8812      JMP [LOOP.T8.7]      ;LOOP ON ERROR IF ENABLED
U 1B40, 09B2.F4 :8813      JMP [NOLOOP.T8.7]      ;GO TO NEXT SECTION IF NOT
                                T8.8:
U 1B42, B6A0.95 :8814      MOV LS[PREVIOUS.ERROR] TO WR[1] ;GET PREVIOUS ERROR NUMBER
                                XOR WR[1] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1B43, 4D82.B5 :8815      DT(LONG)&SET.ALU.CC
U 1B44, 89B2.F9 :8816      JMP.IF[EQL] TO [LOOP.T8.7]      ;LOOP IF SO
                                NOLOOP.T8.7:
U 1B45, 3646.95 :8817      MOV LS[#8] TO WR[1]      ;SET WR1 TO 8
  
```

```

U 1B46, 3E82,95 :8831      MOV WR[1] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO 8
U 1B47, B616,15 :8832      MOV LS[T11] TO WR[0] ;RESTORE WRO
                        XOR WR[0] WITH LS[ONES] TO Q, ;COMPARE WRO WITH ALL ONES
                        DT(LONG)&SET.ALU.CC
U 1B48, 4D9E,35 :8833      SKIP,IF[NEQ] ;SKIP NEXT INSTRUCTION IF ALL BITS IN WRO WERE NOT STILL SET
U 1B49, DB00,01 :8835      JMP [T8.8A] ;GO TO END OF TEST
U 1B4A, 09B5,14 :8836      MOV [LS[ONES]] TO WR[1] ;SET EXPECTED DATA IN WR1
U 1B4B, 369E,95 :8837      JSR [MOV.EX.RE] ;SET UP EXPECTED AND RECEIVED DATA
U 1B4C, 8870,0C :8838      MISC [SET.CP.ATTN] ;REPORT ERROR 8 TO CONSOLE
U 1B4D, 10E0,15 :8839
                        WAIT.T8.8:
U 1B4E, 89B4,E4 :8841      JMP [WAIT.T8.8] ;WAIT FOR CONSOLE TO RESPOND
U 1B4F, 09B2,F4 :8842      JMP [LOOP.T8.7] ;LOOP ON ERROR IF ENABLED
U 1B50, 09B5,44 :8843      JMP [T8.RSTR.LS] ;GO TO NEXT SECTION IF NOT
                        T8.8A:
U 1B51, B6A0,95 :8845      MOV LS[PREVIOUS.ERROR] TO WR[1] ;GET PREVIOUS ERROR NUMBER
                        XOR WR[1] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
                        DT(LONG)&SET.ALU.CC
U 1B52, 4D82,B5 :8847      JMP,IF[EQL] TO [LOOP.T8.7] ;LOOP IF SO
U 1B53, 89B2,F9 :8848
                        T8.RSTR.LS:
U 1B54, 3612,15 :8850      MOV LS[T9] TO WR[0]
U 1B55, BE60,15 :8851      MOV WR[0] TO LS[L30] ;RESTORE LOCATION 30
U 1B56, 3614,15 :8852      MOV LS[T10] TO WR[0]
U 1B57, 3EE6,15 :8853      MOV WR[0] TO LS[L73] ;RESTORE LOCATION 73
                        :8854      END.T8:
  
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

F 16
TEST 9 - 2901 R XNO 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 9 - 2901 R XNOR S Function Test (M8390 CPU Module)

Sequence 200
Page 194

Page "TEST 9 - 2901 R XNOR S Function Test (M8390 CPU Module)"

Test Description:

This test checks the last logical function of the 2901; R XNOR S. The other logical functions have been previously tested. Two micro instructions are needed to test this function. The first is the MCOM LS TO WR instruction. The 2901 is set up for this instruction in the following manner: SRC=D.0 FUNC=R.XNOR.S DST=WRITE.B.F CIN=CIN. The second u-instruction needed is MCOM WR TO WR. In this case the 2901 is set up as follows: ALU=0.A FUNC=R.XNOR.S DST=WRITE.B.F CIN=CIN. These 2 instructions will test the XNOR function for patterns other than XNORing two ones. The hardware (i.e. the ROM and PAL controlling the 2901) does not allow XNOR with ones on both inputs. The data path is as follows: Data from the LS is XNORed with 0 in the 2901 (producing the complement of the original data in LS) and written into a WR[0]. The expected data is put in WR[1] and the error routine is called to compare the data. In the second case (MCOM WR TO WR), WR 0 is loaded from LS and then WR 0 is XNORed with 0 and written back into WR 0. The expected result is then loaded into WR 1 and the error routine is called.

Assumptions:

It is assumed that the error routine will work correctly. All instructions used in that routine have been tested previously. However, since this is the first test that actually uses this routine, Murphy's law may prevail and cause a failure due to the error routine itself (such as a double bit load failure which would not cause a parity error).

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Execute MCOM LS TO WR with data of 0's from LS. Check result for all ones.
- 3) Execute MCOM LS TO WR with data of 1's from LS. Check result for all zeros.
- 4) Execute MCOM LS TO WR with shifting 1's data from LS. Check for correct result (complement of shifting 1's data).
- 5) Repeat step 4 until all 32 bits have been individually tested.
- 6) Repeat steps 2 through 5 with MCOM WR TO WR.

Errors:

- Error 1 - MCOM LS TO WR failed with data of all 0's and all 0's.
Error 2 - MCOM LS TO WR failed with data of all 1's and all 0's.
Error 3 - MCOM LS TO WR failed with data of shifting 1's and all 0's.
Error 4 - MCOM WR TO WR failed with data of all 0's and all 0's.
Error 5 - MCOM WR TO WR failed with data of all 0's and all 1's.
Error 6 - MCOM WR TO WR failed with data of all 0's and shifting 1's.

Debug:

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 9 - 2901 R XNOR S Function Test (M8390 CPU Module)
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
G 16 3-MAR-82 Fiche 1 Frame G16 Sequence 201
Rev 01.00 Page 195

```
:8910  
:8911  
:8912  
:8913  
:8914  
:8915  
:8916  
:8917  
:8918  
:8919  
:8920  
:8921  
:8922  
:8923  
:8924  
:8925  
:8926  
:8927  
:8928  
:8929  
:8930  
:8931  
:8932  
:8933  
:8934  
:8935  
:8936  
:8937  
:8938  
:8939  
U 1B58, B65E,15 :8940  
U 1B59, 3E80,15 :8941  
:8942  
U 1B5A, 10E0,15 :8943  
:8944  
U 1B5B, 09B5,B4 :8945  
U 1B5C, 2F80,15 :8946  
U 1B5D, 3E8A,15 :8947  
U 1B5E, BEA0,15 :8948  
U 1B5F, 2040,15 :8949  
U 1B60, BE82,15 :8950  
U 1B61, 2040,15 :8951  
U 1B62, 3E8C,15 :8952  
:8953  
U 1B63, 2F80,15 :8954  
U 1B64, 5F9C,15 :8955  
U 1B65, 369E,95 :8956  
U 1B66, 0869,3C :8957  
U 1B67, 89B6,34 :8958  
U 1B68, 8870,5C :8959  
:8960  
U 1B69, B69E,15 :8961  
U 1B6A, DF9E,15 :8962  
U 1B6B, 2F82,95 :8963  
U 1B6C, 0869,3C :8964
```

T.9:--
T.9:--
WAIT.T9.0:
LOOP.T9.1:
LOOP.T9.2:

```
MOV LS[BEGIN.TEST] TO WR[0] :SET BIT15 IN WRO FOR CONTROL AND STATUS WORD  
MOV WR[0] TO LS[CONTROL.STATUS] :SET BIT15 IN CNTROL AND STATUS WORD  
: BIT15 INDICATES START OF TEST TO CONSOLE  
MISC [SET.CP.ATTN] :ISSUE CPU ATTN TO CONSOLE  
JMP [WAIT.T9.0] :LOOP HERE WAITING FOR CONSOLE TO RESPOND  
CLR WR[0] :SET WRO TO 0  
MOV WR[0] TO LS[ERROR.MASK] :CLEAR ERROR MASK.  
MOV WR[0] TO LS[PREVIOUS.ERROR] :CLEAR PREVIOUS ERROR NUMBER  
INC WR[0] :SET WRO TO 1  
MOV WR[0] TO LS[ERROR.NUMBER] :SET ERROR NUMBER TO FIRST ERROR  
INC WR[0] :SET WRO TO 2  
MOV WR[0] TO LS[MODULE.NUM] :SET UP MODULE CODE FOR CPU MODULE  
CLR WR[0] :SET ALL BITS IN WRO TO 0  
MCOM LS[ZERO] TO WR[0] :TRY MCOM INS WITH DATA OF ALL 0'S  
MOV LS[ONES] TO WR[1] :SET UP EXPECTED DATA  
JSR [CHECK.RESULT] :CALL ERROR CHECKING SUBROUTINE FOR FIRST TIME  
JMP [LOOP.T9.1] :LOOP ON ERROR IF ENABLED  
JSR [INC.EN] :INCREMENT ERROR NUMBER TO 2  
MOV LS[ONES] TO WR[0] :SET ALL BITS IN WRO  
MCOM LS[ONES] TO WR[0] :TRY MCOM INS WITH DATA OF ALL ONES  
CLR WR[1] :SET UP EXPECTED DATA  
JSR [CHECK.RESULT] :CALL ERROR CHECKING ROUTINE
```


U 1B6D, 89B6,94	:8965	JMP [LOOP.T9.2]	:LOOP ON ERROR IF ENABLED
U 1B6E, 8870,5C	:8966	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 3
U 1B6F, E5F8,15	:8967	CLR LS[OS]	:CLEAR THE OS REGISTER TO PREPARE FOR INDEXING
	:8968		
U 1B70, 2F80,15	:8969	LOOP.T9.3:	
U 1B71, 5FF6,15	:8970	CLR WR[0]	:CLEAR THE WR
U 1B72, 369E,95	:8971	MCOM LS[SHIFT.OS(4-0)] TO WR[0]	:TRY MCOM INS. WITH SHIFTING PATTERN
U 1B73, 45F6,95	:8972	MOV LS[ONES] TO WR[1]	:SET ALL BITS IN WR1
	:8973	BIC LS[SHIFT.OS(4-0)] TO WR[1]	:CLEAR THE BIT THAT WAS USED IN THE MCOM INS
	:8974		:FOR EXPECTED DATA
U 1B74, 0869,3C	:8975	JSR [CHECK.RESULT]	:CALL ERROR CHECKING ROUTINE
U 1B75, 09B7,04	:8976	JMP [LOOP.T9.3]	:LOOP ON ERROR IF ENABLED
U 1B76, B6F8,15	:8977	MOV LS[OS] TO WR[0]	:GET THE OS REGISTER
U 1B77, 2040,15	:8978	INC WR[0]	:INCREMENT THE INDEX
U 1B78, 452C,15	:8979	BIC LS[FFFFFF00] TO WR[0]	:CLEAR HIGH BITS
U 1B79, 3EF8,15	:8980	MOV WR[0] TO LS[OS]	:PUT IT BACK
	:8981	XOR WR[0] WITH LS[#20] TO Q,	:IS IT EQUAL TO 32?
U 1B7A, 4D4A,35	:8982	DT(LONG)&SET.ALU.CC	
U 1B7B, 09B7,01	:8983	JMP.IF[NEQ] TO [LOOP.T9.3]	:GO BACK AND DO NEXT BIT
U 1B7C, 8870,5C	:8984	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 4
	:8985	LOOP.T9.4:	
U 1B7D, 2F80,15	:8986	CLR WR[0]	:SET ALL BITS IN WRO TO 0
U 1B7E, 2F82,95	:8987	CLR WR[1]	:SET ALL BITS IN WRO TO 0
U 1B7F, 20C2,15	:8988	MCOM WR[1] TO WR[0]	:TRY MCOM INS WITH DATA OF ALL 0'S
U 1B80, 369E,95	:8989	MOV LS[ONES] TO WR[1]	:SET UP EXPECTED DATA
U 1B81, 0869,3C	:8990	JSR [CHECK.RESULT]	:CALL ERROR CHECKING SUBROUTINE FOR FIRST TIME
U 1B82, 89B7,D4	:8991	JMP [LOOP.T9.4]	:LOOP ON ERROR IF ENABLED
U 1B83, 8870,5C	:8992	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 5
	:8993	LOOP.T9.5:	
U 1B84, B69E,15	:8994	MOV LS[ONES] TO WR[0]	:SET ALL BITS IN WRO
U 1B85, 369E,95	:8995	MOV LS[ONES] TO WR[1]	:SET ALL BITS IN WR1
U 1B86, 20C2,15	:8996	MCOM WR[1] TO WR[0]	:TRY MCOM INS WITH DATA OF ALL ONES
U 1B87, 2F82,95	:8997	CLR WR[1]	:SET UP EXPECTED DATA
U 1B88, 0869,3C	:8998	JSR [CHECK.RESULT]	:CALL ERROR CHECKING ROUTINE
U 1B89, 89B8,44	:8999	JMP [LOOP.T9.5]	:LOOP ON ERROR IF ENABLED
U 1B8A, 8870,5C	:9000	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 6
U 1B8B, E5F8,15	:9001	CLR LS[OS]	:CLEAR THE OS REGISTER TO PREPARE FOR INDEXING
	:9002	LOOP.T9.6:	
U 1B8C, 2F80,15	:9003	CLR WR[0]	:CLEAR WRO
U 1B8D, B6F6,95	:9004	MOV LS[SHIFT.OS(4-0)] TO WR[1]	:SET ONE BIT IN WR1
U 1B8E, 20C2,15	:9005	MCOM WR[1] TO WR[0]	:TRY MCOM INS. WITH SHIFTING PATTERN
U 1B8F, 369E,95	:9006	MOV LS[ONES] TO WR[1]	:SET ALL BITS IN WR1
U 1B90, 45F6,95	:9007	BIC LS[SHIFT.OS(4-0)] TO WR[1]	:CLEAR THE BIT THAT WAS USED IN THE MCOM INS
	:9008		:FOR EXPECTED DATA
U 1B91, 0869,3C	:9009	JSR [CHECK.RESULT]	:CALL ERROR CHECKING ROUTINE
U 1B92, 09B8,C4	:9010	JMP [LOOP.T9.6]	:LOOP ON ERROR IF ENABLED
U 1B93, B6F8,15	:9011	MOV LS[OS] TO WR[0]	:GET THE OS REGISTER
U 1B94, 2040,15	:9012	INC WR[0]	:INCREMENT THE INDEX
U 1B95, 452C,15	:9013	BIC LS[FFFFFF00] TO WR[0]	:CLEAR HIGH BITS
U 1B96, 3EF8,15	:9014	MOV WR[0] TO LS[OS]	:PUT IT BACK
	:9015	XOR WR[0] WITH LS[#20] TO Q,	:IS IT EQUAL TO 32?
U 1B97, 4D4A,35	:9016	DT(LONG)&SET.ALU.CC	
U 1B98, 09B8,C1	:9017	JMP.IF[NEQ] TO [LOOP.T9.6]	:GO BACK AND DO NEXT BIT

END.T9:

:9018
:9019
:9020
:9021
:9022
:9023
:9024
:9025
:9026
:9027
:9028
:9029
:9030
:9031
:9032
:9033
:9034
:9035
:9036
:9037
:9038
:9039
:9040
:9041
:9042
:9043
:9044
:9045
:9046
:9047
:9048
:9049
:9050
:9051
:9052
:9053
:9054
:9055
:9056
:9057
:9058
:9059
:9060
:9061
:9062
:9063
:9064
:9065
:9066
:9067
:9068
:9069
:9070
:9071
:9072

Page "TEST A - Working Registers Test (M8390 CPU Module)"

++

Test Description:

Only 4 working registers can be accessed directly in the 2901 due to hardware constraints. These are WR 0 through WR 3. WR 5 is used by the DECODE instruction, but it will be tested in a later test of the DECODE functionality. WR 0 and WR 1 have already been tested but will be tested again for simplicity. The data path used is as follows: Data from LS will be written into the 2901 via the D BUS. The data patterns will be all ones, all zeros, shifting 1's and shifting 0's. The output of the internal ALU will be written into the WR being tested and the result checked by an XOR instruction which has been tested previously.

Assumptions:

The error subroutine will not be used for this test because it modifies WR 0 and WR 1, which are being tested. It is assumed that the previous tests of the 2901 have run successfully.

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Write all ones in WR 0 and check the result.
- 3) Write all zeros in WR 0 and check the result.
- 4) Write a shifting 1 and shifting 0 pattern in WR 0 and check the result.
- 5) Repeat steps 2, 3, and 4 for WR 1, 2, and 3.
- 6) Write all 0's in all 4 WR's.
- 7) Read 0's in WR 0, write 1's, and read 1's.
- 8) Go to the next WR and repeat step 7. Repeat this step until all four WR's have been accessed.
- 9) Read 1's in WR 3, write 0's, and read 0's.
- 10) Go to the next WR down and repeat step 9. Repeat this step until WR 0 has been accessed.

Errors:

- Error 1 - WR 0 failed to get written or read properly.
- Error 2 - WR 1 failed to get written or read properly.
- Error 3 - WR 2 failed to get written or read properly.
- Error 4 - WR 3 failed to get written or read properly.
- Error 5 - WR 0 failed march test, ascending addresses.
- Error 6 - WR 1 failed march test, ascending addresses.
- Error 7 - WR 2 failed march test, ascending addresses.
- Error 8 - WR 3 failed march test, ascending addresses.
- Error 9 - WR 3 failed march test, descending addresses.
- Error A - WR 2 failed march test, descending addresses.
- Error B - WR 1 failed march test, descending addresses.
- Error C - WR 0 failed march test, descending addresses.

Debug:

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

J 16
TEST A - Working Re 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST A - Working Registers Test (M8390 CPU Module)

Fiche 1 Frame J16

Sequence 204
Rev 01.00 Page 198

:9073
:9074
:9075
:9076
:9077
:9078
:9079
:9080
:9081
:9082
:9083
:9084
:9085
:9086
:9087
:9088
:9089
:9090
:9091
:9092
:9093
:9094
:9095
:9096
:9097
:9098
:9099
:9100
:9101
:9102
:9103
:9104
:9105
:9106
:9107
:9108
:9109
:9110
:9111
:9112
:9113
:9114
:9115
:9116
:9117
:9118
:9119
:9120
:9121
:9122
:9123
:9124
:9125
:9126
:9127

Error 1 and 2 - It is unlikely that this error will occur since WR 0 and WR 1 have been previously tested. If this error should occur, first run all the previous tests again. If this is the first test to fail, then suspect some kind of timing problem. While looping on error, scope the 2901 looking for the problem. Good luck.

Error 3 and 4 - The most likely problem here is a bad cell in the 2901. Suspect this if 4 or less bits are failing. Another possibility is that the A ADRS lines are failing. The read portion of the test uses the D and A inputs to the 2901 internal ALU as the SRC. The A ADRS is supposed to follow the B ADRS. If it fails to do this, we could be reading a different address than we wrote, causing a failure. Check the A ADRS lines from the A.B ADRS CTL PAL. The A ADRS lines should be the same as the B ADRS lines during a BASIC instruction (the XOR that reads the data).

Error 5 through C - The most likely cause of an error in this area is in the address lines. If the data test ran without failure, a march test failure indicates that the wrong address is being written. For example if WR 3 failed (error 8) it could mean that B ADRS 1 is shorted to B ADRS 0, so when we write address 1, we actually write address 3. Check the address lines at the 2901 first while single stepping and stopping at each instruction that writes the WR. If a problem is found, check the A.B ADRS GEN PAL. The output B ADRS 2 H should always be low while B ADRS 0 and B ADRS 1 H will be dependant on which WR is being written or read. The output should be checked on the write and read u-instructions. The write uses a MOV LS TO WR while the read uses XOR WR WITH LS TO Q. The input to the A.B ADRS GEN PAL for CSR 22 through CSR 17 on the write should be 01 1010. CSR bits 7 and 8 should equal the WR being accessed. The other bits don't matter. For the read (XOR) CSR 22 should be a 1 and CSR bits 7 and 8 should equal the WR being accessed. The other bits don't matter. The 2901 itself could be bad also. Suspect this if only 4 bits or fewer are in error.

--
T.A:

MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
; BIT15 INDICATES START OF TEST TO CONSOLE
MISC [SET.CP.ATTN] ;ISSUE CPU ATTN TO CONSOLE
WAIT.TA.0:
JMP [WAIT.TA.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR WR[0] ;SET WR0 TO 0
MOV WR[0] TO LS[ERROR.MASK] ;CLEAR ERROR MASK.
MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
INC WR[0] ;SET WR0 TO 1
MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
INC WR[0] ;SET WR0 TO 2
MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
MOV LS[ERR.CON] TO WR[0] ;LOAD WR0 WITH BIT 8,AND
MOV WR[0] TO LS[CONTROL.STATUS] ;LOAD CONTROL AND STATUS WORD. BIT 8
; SET INDICATES AN ERROR, THIS WORD IS
; LOADED NOW SO THAT IT NEEDN'T

U 1B99, B65E,15
U 1B9A, 3E80,15
U 1B9B, 10E0,15
U 1B9C, 89B9,C4
U 1B9D, 2F80,15
U 1B9E, 3E8A,15
U 1B9F, BEA0,15
U 1BA0, 2040,15
U 1BA1, BE82,15
U 1BA2, 2040,15
U 1BA3, 3E8C,15
U 1BA4, B63E,15
U 1BA5, 3E80,15

```

U 1BA6, E5F8,15 :9128
U 1BA7, 8870,9C :9129
U 1BA8, B69E,15 :9130
U 1BA9, 4D9E,35 :9131
U 1BAA, DB00,01 :9132
U 1BAB, 89BB,24 :9133
U 1BAC, 369E,95 :9134
U 1BAD, 8870,0C :9135
U 1BAE, 10E0,15 :9136
U 1BAF, 89BA,F4 :9137
U 1BB0, 09BA,74 :9138
U 1BB1, 09BB,54 :9139
U 1BB2, 36A0,15 :9140
U 1BB3, CD82,35 :9141
U 1BB4, 89BA,79 :9142
U 1BB5, 8870,9C :9143
U 1BB6, 369C,15 :9144
U 1BB7, CD9C,35 :9145
U 1BB8, DB00,01 :9146
U 1BB9, 89BC,04 :9147
U 1BBA, B69C,95 :9148
U 1BBB, 8870,0C :9149
U 1BBC, 10E0,15 :9150
U 1BBD, 89BB,D4 :9151
U 1BBE, 09BB,54 :9152
U 1BBF, 89BC,54 :9153
U 1BC0, 36A0,15 :9154
U 1BC1, CD82,35 :9155
U 1BC2, 89BA,79 :9156
U 1BC3, 2F80,15 :9157
U 1BC4, BEA0,15 :9158
U 1BC5, 8870,9C :9159
U 1BC6, 36F6,15 :9160
U 1BC7, CDF6,35 :9161
U 1BC8, DB00,01 :9162
U 1BC9, 09BD,04 :9163
U 1BCA, B6F6,95 :9164
U 1BCB, 8870,0C :9165
U 1BCC, 10E0,15 :9166
U 1BCD, 09BC,D4 :9167
U 1BCE, 89BC,54 :9168
U 1BCF, 89BC,54 :9169
U 1BD0, 36A0,15 :9170
U 1BD1, CD82,35 :9171
U 1BD2, 89BA,79 :9172
U 1BD3, 2F80,15 :9173
U 1BD4, BEA0,15 :9174
U 1BD5, 8870,9C :9175
U 1BD6, 36F6,15 :9176
U 1BD7, CDF6,35 :9177
U 1BD8, DB00,01 :9178
U 1BD9, 09BD,04 :9179
U 1BDA, B6F6,95 :9180
U 1BDB, 8870,0C :9181
U 1BDC, 10E0,15 :9182
U 1BDD, 09BC,D4 :9183
U 1BDE, 89BC,54 :9184
U 1BDF, 89BC,54 :9185
U 1BD0, 36A0,15 :9186
U 1BD1, CD82,35 :9187
U 1BD2, 89BA,79 :9188
U 1BD3, 2F80,15 :9189
U 1BD4, BEA0,15 :9190
U 1BD5, 8870,9C :9191
U 1BD6, 36F6,15 :9192
U 1BD7, CDF6,35 :9193
U 1BD8, DB00,01 :9194
U 1BD9, 09BD,04 :9195
U 1BDA, B6F6,95 :9196
U 1BDB, 8870,0C :9197
U 1BDC, 10E0,15 :9198
U 1BDD, 09BC,D4 :9199
U 1BDE, 89BC,54 :9200
U 1BDF, 89BC,54 :9201
U 1BD0, 36A0,15 :9202
U 1BD1, CD82,35 :9203
U 1BD2, 89BA,79 :9204
U 1BD3, 2F80,15 :9205
U 1BD4, BEA0,15 :9206
U 1BD5, 8870,9C :9207
U 1BD6, 36F6,15 :9208
U 1BD7, CDF6,35 :9209
U 1BD8, DB00,01 :9210
U 1BD9, 09BD,04 :9211
U 1BDA, B6F6,95 :9212
U 1BDB, 8870,0C :9213
U 1BDC, 10E0,15 :9214
U 1BDD, 09BC,D4 :9215
U 1BDE, 89BC,54 :9216
U 1BDF, 89BC,54 :9217
U 1BD0, 36A0,15 :9218
U 1BD1, CD82,35 :9219
U 1BD2, 89BA,79 :9220
U 1BD3, 2F80,15 :9221
U 1BD4, BEA0,15 :9222
U 1BD5, 8870,9C :9223
U 1BD6, 36F6,15 :9224
U 1BD7, CDF6,35 :9225
U 1BD8, DB00,01 :9226
U 1BD9, 09BD,04 :9227
U 1BDA, B6F6,95 :9228
U 1BDB, 8870,0C :9229
U 1BDC, 10E0,15 :9230
U 1BDD, 09BC,D4 :9231
U 1BDE, 89BC,54 :9232
U 1BDF, 89BC,54 :9233
U 1BD0, 36A0,15 :9234
U 1BD1, CD82,35 :9235
U 1BD2, 89BA,79 :9236
U 1BD3, 2F80,15 :9237
U 1BD4, BEA0,15 :9238
U 1BD5, 8870,9C :9239
U 1BD6, 36F6,15 :9240
U 1BD7, CDF6,35 :9241
U 1BD8, DB00,01 :9242
U 1BD9, 09BD,04 :9243
U 1BDA, B6F6,95 :9244
U 1BDB, 8870,0C :9245
U 1BDC, 10E0,15 :9246
U 1BDD, 09BC,D4 :9247
U 1BDE, 89BC,54 :9248
U 1BDF, 89BC,54 :9249
U 1BD0, 36A0,15 :9250
U 1BD1, CD82,35 :9251
U 1BD2, 89BA,79 :9252
U 1BD3, 2F80,15 :9253
U 1BD4, BEA0,15 :9254
U 1BD5, 8870,9C :9255
U 1BD6, 36F6,15 :9256
U 1BD7, CDF6,35 :9257
U 1BD8, DB00,01 :9258
U 1BD9, 09BD,04 :9259
U 1BDA, B6F6,95 :9260
U 1BDB, 8870,0C :9261
U 1BDC, 10E0,15 :9262
U 1BDD, 09BC,D4 :9263
U 1BDE, 89BC,54 :9264
U 1BDF, 89BC,54 :9265
U 1BD0, 36A0,15 :9266
U 1BD1, CD82,35 :9267
U 1BD2, 89BA,79 :9268
U 1BD3, 2F80,15 :9269
U 1BD4, BEA0,15 :9270
U 1BD5, 8870,9C :9271
U 1BD6, 36F6,15 :9272
U 1BD7, CDF6,35 :9273
U 1BD8, DB00,01 :9274
U 1BD9, 09BD,04 :9275
U 1BDA, B6F6,95 :9276
U 1BDB, 8870,0C :9277
U 1BDC, 10E0,15 :9278
U 1BDD, 09BC,D4 :9279
U 1BDE, 89BC,54 :9280
U 1BDF, 89BC,54 :9281
U 1BD0, 36A0,15 :9282
U 1BD1, CD82,35 :9283
U 1BD2, 89BA,79 :9284
U 1BD3, 2F80,15 :9285
U 1BD4, BEA0,15 :9286
U 1BD5, 8870,9C :9287
U 1BD6, 36F6,15 :9288
U 1BD7, CDF6,35 :9289
U 1BD8, DB00,01 :9290
U 1BD9, 09BD,04 :9291
U 1BDA, B6F6,95 :9292
U 1BDB, 8870,0C :9293
U 1BDC, 10E0,15 :9294
U 1BDD, 09BC,D4 :9295
U 1BDE, 89BC,54 :9296
U 1BDF, 89BC,54 :9297
U 1BD0, 36A0,15 :9298
U 1BD1, CD82,35 :9299
U 1BD2, 89BA,79 :9300
U 1BD3, 2F80,15 :9301
U 1BD4, BEA0,15 :9302
U 1BD5, 8870,9C :9303
U 1BD6, 36F6,15 :9304
U 1BD7, CDF6,35 :9305
U 1BD8, DB00,01 :9306
U 1BD9, 09BD,04 :9307
U 1BDA, B6F6,95 :9308
U 1BDB, 8870,0C :9309
U 1BDC, 10E0,15 :9310
U 1BDD, 09BC,D4 :9311
U 1BDE, 89BC,54 :9312
U 1BDF, 89BC,54 :9313
U 1BD0, 36A0,15 :9314
U 1BD1, CD82,35 :9315
U 1BD2, 89BA,79 :9316
U 1BD3, 2F80,15 :9317
U 1BD4, BEA0,15 :9318
U 1BD5, 8870,9C :9319
U 1BD6, 36F6,15 :9320
U 1BD7, CDF6,35 :9321
U 1BD8, DB00,01 :9322
U 1BD9, 09BD,04 :9323
U 1BDA, B6F6,95 :9324
U 1BDB, 8870,0C :9325
U 1BDC, 10E0,15 :9326
U 1BDD, 09BC,D4 :9327
U 1BDE, 89BC,54 :9328
U 1BDF, 89BC,54 :9329
U 1BD0, 36A0,15 :9330
U 1BD1, CD82,35 :9331
U 1BD2, 89BA,79 :9332
U 1BD3, 2F80,15 :9333
U 1BD4, BEA0,15 :9334
U 1BD5, 8870,9C :9335
U 1BD6, 36F6,15 :9336
U 1BD7, CDF6,35 :9337
U 1BD8, DB00,01 :9338
U 1BD9, 09BD,04 :9339
U 1BDA, B6F6,95 :9340
U 1BDB, 8870,0C :9341
U 1BDC, 10E0,15
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST A - Working Registers Test (M8390 CPU Module)
MICRO2 1L(03) 3-MAR-82 09:34:33
L 16
Fiche 1 Frame L16
Sequence 206
Page 200

```
U 1BCE, 89BC,54 :9183      JMP [LOOP.TA.1B]          ;LOOP ON ERROR IF ENABLED
U 1BCF, 09BD,34 :9184      JMP [NOLOOP.TA.1B]       ;GO TO NEXT SECTION IF NOT
:9185
U 1BD0, 36A0,15 :9186      TA.1BB: MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9187      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
:9188      DT(LONG)&SET.ALU.CC
:9189      JMP.IF[EQL] TO [LOOP.TA.1] ;LOOP ON ERROR IF SO
:9190
U 1BD3, 36F8,95 :9191      NOLOOP.TA.1B: MOV LS[OS] TO WR[1] ;GET COUNTER
U 1BD4, 2042,95 :9192      INC WR[1] ;INCREMENT IT
U 1BD5, C52C,95 :9193      BIC LS[FFFFFFF00] TO WR[1] ;CLEAR HIGH BITS
:9194      XOR WR[1] WITH LS[#20] TO Q, ;IS IT EQUAL TO 32?
:9195      DT(LONG)&SET.ALU.CC
:9196      JMP.IF[EQL] TO [TA.1C] ;GO TO NEXT SECTION IF SO
:9197      MOV WR[1] TO LS[OS]
:9198      JMP [TA.1B.0] ;GO DO NEXT BIT
:9199
U 1BDA, E5F8,15 :9200      TA.1C: CLR LS[OS] ;CLEAR THE OS REGISTER
:9201
U 1BDB, 2F80,15 :9202      TA.1C.0: CLR WR[0]
U 1BDC, BEA0,15 :9203      MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR
:9204
U 1BDD, 8870,9C :9205      LOOP.TA.1C: JSR [ISSUE.SS] ;ISSUE SCOPE SYNC
U 1BDE, 5FF6,15 :9206      MCOM LS[SHIFT.OS(4-0)] TO WR[0] ;WRITE SHIFTING 0 PATTERN
U 1BDF, 369E,95 :9207      MOV LS[ONES] TO WR[1] ;SET ALL BITS IN WR1
U 1BE0, 45F6,95 :9208      BIC LS[SHIFT.OS(4-0)] TO WR[1] ;SET UP PROPER CLEARED BIT
:9209      XOR WR[0] WITH WR[1] TO Q, ;CHECK FOR PROPER BIT TO BE CLEARED
:9210      DT(LONG)&SET.ALU.CC
:9211      SKIP.IF[NEQ] ;SKIP NEXT INSTRUCTION IF NOT
:9212      JMP [TA.1CC] ;GO TO NEXT SECTION
:9213
U 1BE1, AB80,B5 :9214      JSR [MOV.EX.RE]
U 1BE2, DB00,01 :9215      MISC [SET.CP.ATTN] ;REPORT ERROR 2 TO CONSOLE
:9216
U 1BE3, 09BE,94 :9217      WAIT.TA.1C: JMP [WAIT.TA.1C] ;WAIT FOR CONSOLE TO RESPOND
U 1BE4, 8870,0C :9218      JMP [LOOP.TA.1C] ;LOOP ON ERROR IF ENABLED
U 1BE5, 10E0,15 :9219      JMP [NOLOOP.TA.1C] ;GO TO NEXT SECTION IF NOT
:9220
U 1BE6, 09BE,64 :9221      TA.1CC: MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
U 1BE7, 89BD,D4 :9222      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1BE8, 09BE,C4 :9223      DT(LONG)&SET.ALU.CC
:9224      JMP.IF[EQL] TO [LOOP.TA.1] ;LOOP ON ERROR IF SO
:9225
U 1BEC, 36F8,95 :9226      NOLOOP.TA.1C: MOV LS[OS] TO WR[1] ;GET COUNTER
U 1BED, 2042,95 :9227      INC WR[1] ;INCREMENT IT
U 1BEE, C52C,95 :9228      BIC LS[FFFFFFF00] TO WR[1] ;CLEAR HIGH BITS
:9229      XOR WR[1] WITH LS[#20] TO Q, ;IS IT EQUAL TO 32?
:9230      DT(LONG)&SET.ALU.CC
:9231      JMP.IF[EQL] TO [TA.2] ;GO TO NEXT SECTION IF SO
:9232      MOV WR[1] TO LS[OS]
:9233      JMP [TA.1C.0] ;GO DO NEXT BIT
:9234
U 1BF3, 8870,5C :9235      TA.2: JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
U 1BF4, E5F8,15 :9236      CLR LS[OS] ;CLEAR THE OS REGISTER
:9237
U 1BF5, 8870,9C :9237      LOOP.TA.2: JSR [ISSUE.SS] ;ISSUE SCOPE SYNC
```



```

U 1BF6, 369E,95 :9238      MOV LS[ONES] TO WR[1]      ;WRITE ONES TO WR1
:9239      XOR WR[1] WITH LS[ONES] TO Q,    ;CHECK FOR ALL BITS TO BE SET
U 1BF7, CD9E,B5 :9240      DT(LONG)&SET.ALU.CC
U 1BF8, DB00,01 :9241      SKIP.IF[NEQ]      ;SKIP NEXT INSTRUCTION IF NOT
U 1BF9, 89C0,14 :9242      JMP [TA.2A]      ;GO TO NEXT SECTION
U 1BFA, 2002,15 :9243      MOV WR[1] TO WR[0]      ;SAVE RECIEVED DATA
U 1BFB, 369E,95 :9244      MOV LS[ONES] TO WR[1]      ;SET UP EXPECTED DATA IN WR1
U 1BFC, 8870,0C :9245      JSR [MOV.EX.RE]
U 1BFD, 10E0,15 :9246      MISC [SET.CP.ATTN]      ;REPORT ERROR 2 TO CONSOLE
:9247
WAIT.TA.2:
U 1BFE, 09BF,E4 :9248      JMP [WAIT.TA.2]      ;WAIT FOR CONSOLE TO REPSOND
U 1BFF, 89BF,54 :9249      JMP [LOOP.TA.2]      ;LOOP ON ERROR IF ENABLED
U 1C00, 89C0,44 :9250      JMP [LOOP.TA.2A]      ;GO TO NEXT SECTION IF NOT
:9251
TA.2A:
U 1C01, 36A0,15 :9252      MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9253      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1C02, CD82,35 :9254      DT(LONG)&SET.ALU.CC
U 1C03, 09BF,59 :9255      JMP.IF[EQ] TO [LOOP.TA.2] ;LOOP ON ERROR IF SO
:9256
LOOP.TA.2A:
U 1C04, 8870,9C :9257      JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC
U 1C05, B69C,95 :9258      MOV LS[ZERO] TO WR[1]      ;WRITE ZERO TO WR1
:9259      XOR WR[1] WITH LS[ZERO] TO Q,      ;CHECK FOR ALL BITS TO BE CLEAR
U 1C06, 4D9C,B5 :9260      DT(LONG)&SET.ALU.CC
U 1C07, DB00,01 :9261      SKIP.IF[NEQ]      ;SKIP NEXT INSTRUCTION IF NOT
U 1C08, 89C1,04 :9262      JMP [TA.2B]      ;GO TO NEXT SECTION
U 1C09, 2002,15 :9263      MOV WR[1] TO WR[0]      ;SAVE RECIEVED DATA
U 1C0A, B69C,95 :9264      MOV LS[ZERO] TO WR[1]      ;SET UP EXPECTED DATA IN WR1
U 1C0B, 8870,0C :9265      JSR [MOV.EX.RE]
U 1C0C, 10E0,15 :9266      MISC [SET.CP.ATTN]      ;REPORT ERROR 2 TO CONSOLE
:9267
WAIT.TA.2A:
U 1C0D, 89C0,D4 :9268      JMP [WAIT.TA.2A]      ;WAIT FOR CONSOLE TO REPSOND
U 1C0E, 89C0,44 :9269      JMP [LOOP.TA.2A]      ;LOOP ON ERROR IF ENABLED
U 1C0F, 89C1,54 :9270      JMP [LOOP.TA.2B]      ;GO TO NEXT SECTION IF NOT
:9271
TA.2B:
U 1C10, 36A0,15 :9272      MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9273      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1C11, CD82,35 :9274      DT(LONG)&SET.ALU.CC
U 1C12, 09BF,59 :9275      JMP.IF[EQ] TO [LOOP.TA.2] ;LOOP ON ERROR IF SO
:9276
TA.2B.0:
U 1C13, 2F80,15 :9277      CLR WR[0]
U 1C14, BEA0,15 :9278      MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR
:9279
LOOP.TA.2B:
U 1C15, 8870,9C :9280      JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC
U 1C16, B6F6,95 :9281      MOV LS[SHIFT.OS(4-0)] TO WR[1] ;WRITE SHIFTING 1 PATTERN
:9282      XOR WR[1] WITH LS[SHIFT.OS(4-0)] TO Q, ;CHECK FOR PROPER BIT TO BE SET
U 1C17, 4DF6,B5 :9283      DT(LONG)&SET.ALU.CC
U 1C18, DB00,01 :9284      SKIP.IF[NEQ]      ;SKIP NEXT INSTRUCTION IF NOT
U 1C19, 09C2,14 :9285      JMP [TA.2BB]      ;GO TO NEXT SECTION
U 1C1A, 2002,15 :9286      MOV WR[1] TO WR[0]      ;SAVE RECIEVED DATA
U 1C1B, B6F6,95 :9287      MOV LS[SHIFT.OS(4-0)] TO WR[1] ;SET UP EXPECTED DATA IN WR1
U 1C1C, 8870,0C :9288      JSR [MOV.EX.RE]
U 1C1D, 10E0,15 :9289      MISC [SET.CP.ATTN]      ;REPORT ERROR 2 TO CONSOLE
:9290
WAIT.TA.2B:
U 1C1E, 09C1,E4 :9291      JMP [WAIT.TA.2B]      ;WAIT FOR CONSOLE TO REPSOND
U 1C1F, 89C1,54 :9292      JMP [LOOP.TA.2B]      ;LOOP ON ERROR IF ENABLED

```



```

U 1C48, 369F,15 :9348      MOV LS[ONES] TO WR[2]      ;WRITE ONES TO WR2
:9349      XOR WR[2] WITH LS[ONES] TO Q, ;CHECK FOR ALL BITS TO BE SET
U 1C49, CD9F,35 :9350      DT(LONG)&SET.ALU.CC
U 1C4A, DB00,01 :9351      SKIP,IF[NEQ]      ;SKIP NEXT INSTRUCTION IF NOT
U 1C4B, 09C5,34 :9352      JMP [TA,3A]      ;GO TO NEXT SECTION
U 1C4C, 2004,15 :9353      MOV WR[2] TO WR[0]      ;SET UP RECIEVED DATA IN WR0
U 1C4D, 369E,95 :9354      MOV LS[ONES] TO WR[1]      ;SET UP EXPECTED DATA IN WR1
U 1C4E, 8870,0C :9355      JSR [MOV.EX.RE]
U 1C4F, 10E0,15 :9356      MISC [SET.CP.ATTN]      ;REPORT ERROR 3 TO CONSOLE
:9357      WAIT.TA.3:
U 1C50, 09C5,04 :9358      JMP [WAIT.TA.3]      ;WAIT FOR CONSOLE TO REPSOND
U 1C51, 09C4,74 :9359      JMP [LOOP.TA.3]      ;LOOP ON ERROR IF ENABLED
U 1C52, 09C5,64 :9360      JMP [LOOP.TA.3A]      ;GO TO NEXT SECTION IF NOT
:9361      TA.3A:
U 1C53, 36A0,15 :9362      MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9363      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1C54, CD82,35 :9364      DT(LONG)&SET.ALU.CC
U 1C55, 89C4,79 :9365      JMP,IF[EQL] TO [LOOP.TA.3] ;LOOP ON ERROR IF SO
:9366      LOOP.TA.3A:
U 1C56, 8870,9C :9367      JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC
U 1C57, B69D,15 :9368      MOV LS[ZERO] TO WR[2]      ;WRITE ZERO TO WR2
:9369      XOR WR[2] WITH LS[ZERO] TO Q, ;CHECK FOR ALL BITS TO BE CLEAR
U 1C58, 4D9D,35 :9370      DT(LONG)&SET.ALU.CC
U 1C59, DB00,01 :9371      SKIP,IF[NEQ]      ;SKIP NEXT INSTRUCTION IF NOT
U 1C5A, 89C6,24 :9372      JMP [TA,3B]      ;GO TO NEXT SECTION
U 1C5B, 2004,15 :9373      MOV WR[2] TO WR[0]      ;SET UP RECEIVED DATA IN WR0
U 1C5C, B69C,95 :9374      MOV LS[ZERO] TO WR[1]      ;SET UP EXPECTED DATA IN WR1
U 1C5D, 8870,0C :9375      JSR [MOV.EX.RE]
U 1C5E, 10E0,15 :9376      MISC [SET.CP.ATTN]      ;REPORT ERROR 3 TO CONSOLE
:9377      WAIT.TA.3A:
U 1C5F, 09C5,F4 :9378      JMP [WAIT.TA.3A]      ;WAIT FOR CONSOLE TO REPSOND
U 1C60, 09C5,64 :9379      JMP [LOOP.TA.3A]      ;LOOP ON ERROR IF ENABLED
U 1C61, 89C6,74 :9380      JMP [LOOP.TA.3B]      ;GO TO NEXT SECTION IF NOT
:9381      TA.3B:
U 1C62, 36A0,15 :9382      MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9383      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1C63, CD82,35 :9384      DT(LONG)&SET.ALU.CC
U 1C64, 89C4,79 :9385      JMP,IF[EQL] TO [LOOP.TA.3] ;LOOP ON ERROR IF SO
:9386      TA.3B.0:
U 1C65, 2F80,15 :9387      CLR WR[0]
U 1C66, BEA0,15 :9388      MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR
:9389      LOOP.TA.3B:
U 1C67, 8870,9C :9390      JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC
U 1C68, B6F7,15 :9391      MOV LS[SHIFT.OS(4-0)] TO WR[2] ;WRITE SHIFTING 1 PATTERN
:9392      XOR WR[2] WITH LS[SHIFT.OS(4-0)] TO Q, ;CHECK FOR PROPER BIT TO BE SET
U 1C69, 4DF7,35 :9393      DT(LONG)&SET.ALU.CC
U 1C6A, DB00,01 :9394      SKIP,IF[NEQ]      ;SKIP NEXT INSTRUCTION IF NOT
U 1C6B, 89C7,34 :9395      JMP [TA,3BB]      ;GO TO NEXT SECTION
U 1C6C, 2004,15 :9396      MOV WR[2] TO WR[0]      ;SET UP RECEIVED DATA IN WR0
U 1C6D, B6F6,95 :9397      MOV LS[SHIFT.OS(4-0)] TO WR[1] ;SET UP EXPECTED DATA IN WR1
U 1C6E, 8870,0C :9398      JSR [MOV.EX.RE]
U 1C6F, 10E0,15 :9399      MISC [SET.CP.ATTN]      ;REPORT ERROR 3 TO CONSOLE
:9400      WAIT.TA.3B:
U 1C70, 89C7,04 :9401      JMP [WAIT.TA.3B]      ;WAIT FOR CONSOLE TO REPSOND
U 1C71, 89C6,74 :9402      JMP [LOOP.TA.3B]      ;LOOP ON ERROR IF ENABLED
    
```


ZZ-ENKCA-1.0	:	ENKCA.MIC							
: ENKCA.MCR			MICRO2 1L(03)	TEST A - Working Re	3-MAR-82	Fiche 2	Frame E1	Sequence 210	
: ENKCA.MIC			TEST A - Working Registers Test (M8390	CPU Module)	ENKCA Micro-diagnostic for DAP module	Rev 01.00	Page	204	

U 1C72, 89C7,64	:9403		JMP [NOLOOP.TA.3B]	:GO TO NEXT SECTION IF NOT
U 1C73, 36A0,15	:9404	TA.3BB:	MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR NUMBER
	:9405		XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
U 1C74, CD82,35	:9406		DT(LONG)&SET.ALU.CC	
U 1C75, 89C4,79	:9407		JMP.IF[EQL] TO [LOOP.TA.3]	:LOOP ON ERROR IF SO
	:9408	NOLOOP.TA.3B:		
	:9409		MOV LS[OS] TO WR[1]	:GET COUNTER
U 1C76, 36F8,95	:9410		INC WR[1]	: INCREMENT IT
U 1C77, 2042,95	:9411		BIC LS[FFFFFFF0] TO WR[1]	: CLEAR HIGH BITS
U 1C78, C52C,95	:9412		MOV WR[1] TO LS[OS]	:PUT IT BACK
U 1C79, BEF8,95	:9413		XOR WR[1] WITH LS[#20] TO Q,	:IS IT EQUAL TO 32?
	:9414		DT(LONG)&SET.ALU.CC	
U 1C7A, CD4A,B5	:9415		JMP.IF[EQL] TO [TA.3C]	:GO TO NEXT SECTION IF SO
U 1C7B, 89C7,E9	:9416		MOV WR[1] TO LS[OS]	
U 1C7C, BEF8,95	:9417		JMP [TA.3B.0]	:GO DO NEXT BIT
U 1C7D, 09C6,54	:9418	TA.3C:		
	:9419		CLR LS[OS]	:CLEAR THE OS REGISTER
U 1C7E, E5F8,15	:9420	TA.3C.0:		
	:9421		CLR WR[0]	
U 1C7F, 2F80,15	:9422		MOV WR[0] TO LS[PREVIOUS.ERROR]	:CLEAR PREVIOUS ERROR
U 1C80, BEA0,15	:9423	LOOP.TA.3C:		
	:9424		JSR [ISSUE.SS]	:ISSUE SCOPE SYNC
U 1C81, 8870,9C	:9425		MCOM LS[SHIFT.OS(4-0)] TO WR[2]	:WRITE SHIFTING 0 PATTERN
U 1C82, DFF7,15	:9426		MOV LS[ONES] TO WR[1]	:SET ALL BITS IN WR1
U 1C83, 369E,95	:9427		BIC LS[SHIFT.OS(4-0)] TO WR[1]	:SET UP PROPER CLEARED BIT
U 1C84, 45F6,95	:9428		XOR WR[2] WITH WR[1] TO Q,	:CHECK FOR PROPER BIT TO BE CLEARED
	:9429		DT(LONG)&SET.ALU.CC	
U 1C85, 2B84,B5	:9430		SKIP.IF[NEQ]	:SKIP NEXT INSTRUCTION IF NOT
U 1C86, DB00,01	:9431		JMP [TA.3CC]	:GO TO NEXT SECTION
U 1C87, 09C8,E4	:9432		MOV WR[2] TO WR[0]	:SET UP RECEIVED DATA IN WRO
U 1C88, 2004,15	:9433		JSR [MOV.EX.RE]	
U 1C89, 8870,0C	:9434		MISC [SET.CP.ATTN]	:REPORT ERROR 3 TO CONSOLE
U 1C8A, 10E0,15	:9435	WAIT.TA.3C:		
	:9436		JMP [WAIT.TA.3C]	:WAIT FOR CONSOLE TO RESPOND
U 1C8B, 09C8,B4	:9437		JMP [LOOP.TA.3C]	:LOOP ON ERROR IF ENABLED
U 1C8C, 09C8,14	:9438		JMP [NOLOOP.TA.3C]	:GO TO NEXT SECTION IF NOT
U 1C8D, 89C9,14	:9439	TA.3CC:		
	:9440		MOV LS[PREVIOUS.ERROR] TO WR[0]	:GET PREVIOUS ERROR NUMBER
U 1C8E, 36A0,15	:9441		XOR WR[0] WITH LS[ERROR.NUMBER]	TO Q, :CURRENT ERROR?
	:9442		DT(LONG)&SET.ALU.CC	
U 1C8F, CD82,35	:9443		JMP.IF[EQL] TO [LOOP.TA.3]	:LOOP ON ERROR IF SO
U 1C90, 89C4,79	:9444	NOLOOP.TA.3C:		
	:9445		MOV LS[OS] TO WR[1]	:GET COUNTER
U 1C91, 36F8,95	:9446		INC WR[1]	: INCREMENT IT
U 1C92, 2042,95	:9447		BIC LS[FFFFFFF0] TO WR[1]	: CLEAR HIGH BITS
U 1C93, C52C,95	:9448		MOV WR[1] TO LS[OS]	:PUT IT BACK
U 1C94, BEF8,95	:9449		XOR WR[1] WITH LS[#20] TO Q,	:IS IT EQUAL TO 32?
	:9450		DT(LONG)&SET.ALU.CC	
U 1C95, CD4A,B5	:9451		JMP.IF[EQL] TO [TA.4]	:GO TO NEXT SECTION IF SO
U 1C96, 89C9,99	:9452		MOV WR[1] TO LS[OS]	
U 1C97, BEF8,95	:9453	TA.4:		
U 1C98, 89C7,F4	:9454		JMP [TA.3C.0]	:GO DO NEXT BIT
	:9455		JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 4
U 1C99, 8870,5C	:9456		CLR LS[OS]	:CLEAR THE OS REGISTER
U 1C9A, E5F8,15	:9457			

```

:9458 LOOP.TA.4:
U 1C9B, 8870,9C :9459 JSR [ISSUE.SS] :ISSUE SCOPE SYNC
U 1C9C, B69F,95 :9460 MOV LS[ONES] TO WR[3] :WRITE ONES TO WR3
:9461 XOR WR[3] WITH LS[ONES] TO Q, :CHECK FOR ALL BITS TO BE SET
U 1C9D, 4D9F,B5 :9462 DT(LONG)&SET.ALU.CC
U 1C9E, DB00,01 :9463 SKIP.IF[NEQ] :SKIP NEXT INSTRUCTION IF NOT
U 1C9F, 89CA,74 :9464 JMP [TA.4A] :GO TO NEXT SECTION
U 1CA0, A006,15 :9465 MOV WR[3] TO WR[0] :SET UP RECEIVED DATA IN WRO
U 1CA1, 369E,95 :9466 MOV LS[ONES] TO WR[1] :SET UP EXPECTED DATA IN WR1
U 1CA2, 8870,0C :9467 JSR [MOV.EX.RE]
U 1CA3, 10E0,15 :9468 MISC [SET.CP.ATTN] :REPORT ERROR 4 TO CONSOLE
:9469
U 1CA4, 89CA,44 :9470 JMP [WAIT.TA.4] :WAIT FOR CONSOLE TO REPSOND
U 1CA5, 89C9,B4 :9471 JMP [LOOP.TA.4] :LOOP ON ERROR IF ENABLED
U 1CA6, 09CA,A4 :9472 JMP [LOOP.TA.4A] :GO TO NEXT SECTION IF NOT
:9473
U 1CA7, 36A0,15 :9474 TA.4A: MOV LS[PREVIOUS.ERROR] TO WR[0] :GET PREVIOUS ERROR NUMBER
:9475 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, :CURRENT ERROR?
U 1CA8, CD82,35 :9476 DT(LONG)&SET.ALU.CC
U 1CA9, 09C9,B9 :9477 JMP.IF[EQL] TO [LOOP.TA.4] :LOOP ON ERROR IF SO
:9478
U 1CAA, 8870,9C :9479 LOOP.TA.4A: JSR [ISSUE.SS] :ISSUE SCOPE SYNC
U 1CAB, 369D,95 :9480 MOV LS[ZERO] TO WR[3] :WRITE ZERO TO WR3
:9481 XOR WR[3] WITH LS[ZERO] TO Q, :CHECK FOR ALL BITS TO BE CLEAR
U 1CAC, CD9D,B5 :9482 DT(LONG)&SET.ALU.CC
U 1CAD, DB00,01 :9483 SKIP.IF[NEQ] :SKIP NEXT INSTRUCTION IF NOT
U 1CAE, 89CB,64 :9484 JMP [TA.4B] :GO TO NEXT SECTION
U 1CAF, A006,15 :9485 MOV WR[3] TO WR[0] :SET UP RECEIVED DATA IN WRO
U 1CB0, B69C,95 :9486 MOV LS[ZERO] TO WR[1] :SET UP EXPECTED DATA IN WR1
U 1CB1, 8870,0C :9487 JSR [MOV.EX.RE]
U 1CB2, 10E0,15 :9488 MISC [SET.CP.ATTN] :REPORT ERROR 4 TO CONSOLE
:9489
U 1CB3, 89CB,34 :9490 WAIT.TA.4A: JMP [WAIT.TA.4A] :WAIT FOR CONSOLE TO REPSOND
U 1CB4, 09CA,A4 :9491 JMP [LOOP.TA.4A] :LOOP ON ERROR IF ENABLED
U 1CB5, 09CB,B4 :9492 JMP [LOOP.TA.4B] :GO TO NEXT SECTION IF NOT
:9493
U 1CB6, 36A0,15 :9494 TA.4B: MOV LS[PREVIOUS.ERROR] TO WR[0] :GET PREVIOUS ERROR NMBER
:9495 XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, :CURRENT ERROR?
U 1CB7, CD82,35 :9496 DT(LONG)&SET.ALU.CC
U 1CB8, 09C9,B9 :9497 JMP.IF[EQL] TO [LOOP.TA.4] :LOOP ON ERROR IF SO
:9498
U 1CB9, 2F80,15 :9499 TA.4B.0: CLR WR[0]
U 1CBA, BEA0,15 :9500 MOV WR[0] TO LS[PREVIOUS.ERROR] :CLEAR PREVIOUS ERROR
:9501
U 1CBB, 8870,9C :9502 LOOP.TA.4B: JSR [ISSUE.SS] :ISSUE SCOPE SYNC
U 1CBC, 36F7,95 :9503 MOV LS[SHIFT.OS(4-0)] TO WR[3] :WRITE SHIFTING 1 PATTERN
:9504 XOR WR[3] WITH LS[SHIFT.OS(4-0)] TO Q, :CHECK FOR PROPER BIT TO BE SET
U 1CBD, CDF7,B5 :9505 DT(LONG)&SET.ALU.CC
U 1CBE, DB00,01 :9506 SKIP.IF[NEQ] :SKIP NEXT INSTRUCTION IF NOT
U 1CBF, 89CC,74 :9507 JMP [TA.4BB] :GO TO NEXT SECTION
U 1CC0, A006,15 :9508 MOV WR[3] TO WR[0] :SET UP RECEIVED DATA IN WRO
U 1CC1, B6F6,95 :9509 MOV LS[SHIFT.OS(4-0)] TO WR[1] :SET UP EXPECTED DATA IN WR1
U 1CC2, 8870,0C :9510 JSR [MOV.EX.RE]
U 1CC3, 10E0,15 :9511 MISC [SET.CP.ATTN] :REPORT ERROR 4 TO CONSOLE
:9512
WAIT.TA.4B:
  
```


ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

TEST A - Working Re 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module Rev 01.00
TEST A - Working Registers Test (M8390 CPU Module)

Fiche 2 Frame G1

Sequence 212

Page 206

```
U 1CC4, 89CC,44 :9513      JMP [WAIT.TA.4B]      ;WAIT FOR CONSOLE TO RESPOND
U 1CC5, 09CB,B4 :9514      JMP [LOOP.TA.4B]      ;LOOP ON ERROR IF ENABLED
U 1CC6, 09CC,A4 :9515      JMP [NOLoop.TA.4B]    ;GO TO NEXT SECTION IF NOT
:9516
U 1CC7, 36A0,15 :9517      TA.4BB: MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9518      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1CC8, CD82,35 :9519      DT(LONG)&SET.ALU.CC
U 1CC9, 09C9,B9 :9520      JMP.IF[EQL] TO [LOOP.TA.4] ;LOOP ON ERROR IF SO
:9521
U 1CCA, 36F8,95 :9522      NOLoop.TA.4B: MOV LS[OS] TO WR[1] ;GET COUNTER
U 1CCB, 2042,95 :9523      INC WR[1] ;INCREMENT IT
U 1CCC, C52C,95 :9524      BIC LS[FFFFFFF00] TO WR[1] ;CLEAR HIGH BITS
U 1CCD, BEF8,95 :9525      MOV WR[1] TO LS[OS] ;PUT IT BACK
:9526      XOR WR[1] WITH LS[#20] TO Q, ;IS IT EQUAL TO 32?
U 1CCE, CD4A,B5 :9527      DT(LONG)&SET.ALU.CC
U 1CCF, 89CD,29 :9528      JMP.IF[EQL] TO [TA.4C] ;GO TO NEXT SECTION IF SO
U 1CD0, BEF8,95 :9529      MOV WR[1] TO LS[OS]
U 1CD1, 89CB,94 :9530      JMP [TA.4B.0] ;GO DO NEXT BIT
:9531
U 1CD2, E5F8,15 :9532      TA.4C: CLR LS[OS] ;CLEAR THE OS REGISTER
:9533
U 1CD3, 2F80,15 :9534      TA.4C.0: CLR WR[0]
U 1CD4, BEA0,15 :9535      MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR
:9536
U 1CD5, 8870,9C :9537      LOOP.TA.4C: JSR [ISSUE.SS] ;ISSUE SCOPE SYNC
U 1CD6, 5FF7,95 :9538      MCOM LS[SHIFT.OS(4-0)] TO WR[3] ;WRITE SHIFTING 0 PATTERN
U 1CD7, 369E,95 :9539      MOV LS[ONES] TO WR[1] ;SET ALL BITS IN WR1
U 1CD8, 45F6,95 :9540      BIC LS[SHIFT.OS(4-0)] TO WR[1] ;SET UP PROPER CLEARED BIT
:9541      XOR WR[3] WITH WR[1] TO Q, ;CHECK FOR PROPER BIT TO BE CLEARED
U 1CD9, AB86,B5 :9542      DT(LONG)&SET.ALU.CC
U 1CDA, DB00,01 :9543      SKIP.IF[NEQ] ;SKIP NEXT INSTRUCTION IF NOT
U 1CDB, 09CE,24 :9544      JMP [TA.4CC] ;GO TO NEXT SECTION
U 1CDC, A006,15 :9545      MOV WR[3] TO WR[0] ;SET UP RECEIVED DATA IN WRO
U 1CDD, 8870,0C :9546      JSR [MOV.EX.RE]
U 1CDE, 10E0,15 :9547      MISC [SET.CP.ATTN] ;REPORT ERROR 4 TO CONSOLE
:9548
U 1CDF, 89CD,F4 :9549      WAIT.TA.4C: JMP [WAIT.TA.4C] ;WAIT FOR CONSOLE TO RESPOND
U 1CE0, 89CD,54 :9550      JMP [LOOP.TA.4C] ;LOOP ON ERROR IF ENABLED
U 1CE1, 89CE,54 :9551      JMP [NOLoop.TA.4C] ;GO TO NEXT SECTION IF NOT
:9552
U 1CE2, 36A0,15 :9553      TA.4CC: MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9554      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1CE3, CD82,35 :9555      DT(LONG)&SET.ALU.CC
U 1CE4, 09C9,B9 :9556      JMP.IF[EQL] TO [LOOP.TA.4] ;LOOP ON ERROR IF SO
:9557
U 1CE5, 36F8,95 :9558      NOLoop.TA.4C: MOV LS[OS] TO WR[1] ;GET COUNTER
U 1CE6, 2042,95 :9559      INC WR[1] ;INCREMENT IT
U 1CE7, C52C,95 :9560      BIC LS[FFFFFFF00] TO WR[1] ;CLEAR HIGH BITS
U 1CE8, BEF8,95 :9561      MOV WR[1] TO LS[OS] ;PUT IT BACK
:9562      XOR WR[1] WITH LS[#20] TO Q, ;IS IT EQUAL TO 32?
U 1CE9, CD4A,B5 :9563      DT(LONG)&SET.ALU.CC
U 1CEA, 89CE,D9 :9564      JMP.IF[EQL] TO [TA.5] ;GO TO NEXT SECTION IF SO
U 1CEB, BEF8,95 :9565      MOV WR[1] TO LS[OS]
U 1CEC, 89CD,34 :9566      JMP [TA.4C.0] ;GO DO NEXT BIT
:9567
U 1CEC, 89CD,34 :9567      TA.5:
```



```

U 1CED, 8870,5C :9568      JSR [INC.EN]          :INCREMENT ERROR NUMBER TO 5
U 1CEE, 369C,15 :9569      MOV LS[ZERO] TO WR[0]       :CLEAR ALL BITS IN WR0
U 1CEF, B69C,95 :9570      MOV LS[ZERO] TO WR[1]       :CLEAR ALL BITS IN WR1
U 1CF0, B69D,15 :9571      MOV LS[ZERO] TO WR[2]       :CLEAR ALL BITS IN WR2
U 1CF1, 369D,95 :9572      MOV LS[ZERO] TO WR[3]       :CLEAR ALL BITS IN WR3
:9573
LOOP.TA.5:
U 1CF2, 8870,9C :9574      JSR [ISSUE.SS]          :ISSUE SCOPE SYNC
:9575      XOR WR[0] WITH LS[ZERO] TO Q,       :READ 0'S
:9576      DT(LONG)&SET.ALU.CC
U 1CF3, CD9C,35 :9576      CLR WR[1]
:9577      :SET UP EXPECTED DATA IN WR1
U 1CF4, 2F82,95 :9577      JMP.IF[NEQ] TO [TA.5.ERR]       :REPORT ERROR IF WR0 NOT CLEAR
U 1CF5, 89CF,B1 :9578      MOV LS[ONES] TO WR[0]
:9579      :WRITE 1'S
U 1CF6, B69E,15 :9579      XOR WR[0] WITH LS[ONES] TO Q,   :READ 1'S
:9580      DT(LONG)&SET.ALU.CC
U 1CF7, 4D9E,35 :9581      SKIP.IF[NEQ]
:9582      :SKIP NEXT INSTRUCTION IF ALL BITS NOT SET
U 1CF8, DB00,01 :9582      JMP [TA.6]
:9583      :GO TO NEXT SECTION
U 1CF9, 89D0,04 :9583      MOV LS[ONES] TO WR[1]
:9584      :SET UP EXPECTED DATA IN WR1
:9585
TA.5.ERR:
U 1CFB, 8870,0C :9586      JSR [MOV.EX.RE]
:9587      MISC [SET.CP.ATTN]
:9588      :REPORT ERROR 5 TO CONSOLE
WAIT.TA.5:
U 1CFD, 89CF,D4 :9589      JMP [WAIT.TA.5]
:9590      :WAIT FOR CONSOLE TO RESPOND
U 1CFE, 89CF,24 :9590      JMP [LOOP.TA.5]
:9591      :LOOP ON ERROR IF ENABLED
U 1CFF, 89D0,34 :9591      JMP [NOLOOP.TA.5]
:9592      :GO TO NEXT SECTION IF NOT
:9592
TA.6:
U 1D00, 36A0,15 :9593      MOV LS[PREVIOUS.ERROR] TO WR[0] :GET PREVIOUS ERROR NUMBER
:9594      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, :CURRENT ERROR?
:9595      DT(LONG)&SET.ALU.CC
U 1D01, CD82,35 :9595      JMP.IF[EQL] TO [LOOP.TA.5]
:9596      :LOOP ON ERROR IF SO
U 1D02, 09CF,29 :9596      :NOLOOP.TA.5:
:9597      JSR [INC.EN]
:9598      :INCREMENT ERROR NUMBER TO 6
:9599
LOOP.TA.6:
U 1D04, 8870,9C :9600      JSR [ISSUE.SS]          :ISSUE SCOPE SYNC
:9601      XOR WR[1] WITH LS[ZERO] TO Q,       :READ 0'S
:9602      DT(LONG)&SET.ALU.CC
U 1D05, 4D9C,B5 :9602      MOV WR[1] TO WR[0]
:9603      :SET UP RECEIVED DATA IN WR0
U 1D06, 2002,15 :9603      CLR WR[1]
:9604      :SET UP EXPECTED DATA IN WR1
U 1D07, 2F82,95 :9604      JMP.IF[NEQ] TO [TA.6.ERR]       :REPORT ERROR IF WR0 NOT CLEAR
U 1D08, 89D0,F1 :9605      MOV LS[ONES] TO WR[1]
:9606      :WRITE 1'S
U 1D09, 369E,95 :9606      XOR WR[1] WITH LS[ONES] TO Q,   :READ 1'S
:9607      DT(LONG)&SET.ALU.CC
U 1DOA, CD9E,B5 :9608      SKIP.IF[NEQ]
:9609      :SKIP NEXT INSTRUCTION IF ALL BITS NOT SET
U 1DOB, DB00,01 :9609      JMP [TA.7]
:9610      :GO TO NEXT SECTION
U 1DOC, 89D1,44 :9610      MOV WR[1] TO WR[0]
:9611      :SET UP RECEIVED DATA IN WR0
U 1DOD, 2002,15 :9611      MOV LS[ONES] TO WR[1]
:9612      :SET UP EXPECTED DATA IN WR1
:9613
TA.6.ERR:
U 1DOF, 8870,0C :9614      JSR [MOV.EX.RE]
:9615      MISC [SET.CP.ATTN]
:9616      :REPORT ERROR 6 TO CONSOLE
WAIT.TA.6:
U 1D11, 89D1,14 :9617      JMP [WAIT.TA.6]
:9618      :WAIT FOR CONSOLE TO RESPOND
U 1D12, 09D0,44 :9618      JMP [LOOP.TA.6]
:9619      :LOOP ON ERROR IF ENABLED
U 1D13, 89D1,74 :9619      JMP [NOLOOP.TA.6]
:9620      :GO TO NEXT SECTION IF NOT
:9620
TA.7:
U 1D14, 36A0,15 :9621      MOV LS[PREVIOUS.ERROR] TO WR[0] :GET PREVIOUS ERROR NUMBER
:9622      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, :CURRENT ERROR?
    
```

```

U 1D15, CD82.35 :9623      DT(LONG)&SET.ALU.CC
U 1D16, 89D0.49 :9624      JMP.IF[EQ] TO [LOOP.TA.6]      ;LOOP ON ERROR IF SO
                                NOLOOP.TA.6:
                                JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 7
U 1D17, 8870.5C :9625      LOOP.TA.7:
                                JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC
                                XOR WR[2] WITH LS[ZERO] TO Q, ;READ 0'S
U 1D18, 8870.9C :9626      DT(LONG)&SET.ALU.CC
U 1D19, 4D9D.35 :9627      MOV WR[2] TO WR[0]      ;SET UP RECEIVED DATA IN WRO
U 1D1A, 2004.15 :9628      CLR WR[1]      ;SET UP EXPECTED DATA IN WR1
U 1D1B, 2F82.95 :9629      JMP.IF[NEQ] TO [TA.7.ERR] ;REPORT ERROR IF WRO NOT CLEAR
U 1D1C, 09D2.31 :9630      MOV LS[ONES] TO WR[2] ;WRITE 1'S
U 1D1D, 369F.15 :9631      XOR WR[2] WITH LS[ONES] TO Q, ;READ 1'S
                                DT(LONG)&SET.ALU.CC
U 1D1E, CD9F.35 :9632      SKIP.IF[NEQ]      ;SKIP NEXT INSTRUCTION IF ALL BITS NOT SET
U 1D1F, DB00.01 :9633      JMP [TA.8]      ;GO TO NEXT SECTION
U 1D20, 89D2.84 :9634      MOV WR[2] TO WR[0]      ;SET UP RECEIVED DATA IN WRO
U 1D21, 2004.15 :9635      MOV LS[ONES] TO WR[1] ;SET UP EXPECTED DATA IN WR1
U 1D22, 369E.95 :9636      TA.7.ERR:
                                JSR [MOV.EX.RE]
                                MISC [SET.CP.ATTN] ;REPORT ERROR 7 TO CONSOLE
U 1D23, 8870.0C :9637      WAIT.TA.7:
                                JMP [WAIT.TA.7] ;WAIT FOR CONSOLE TO RESPOND
U 1D24, 10E0.15 :9638      JMP [LOOP.TA.7] ;LOOP ON ERROR IF ENABLED
U 1D25, 09D2.54 :9639      JMP [NOLOOP.TA.7] ;GO TO NEXT SECTION IF NOT
U 1D26, 89D1.84 :9640      TA.8:
                                MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
U 1D27, 89D2.84 :9641      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
                                DT(LONG)&SET.ALU.CC
U 1D28, 36A0.15 :9642      JMP.IF[EQ] TO [LOOP.TA.7] ;LOOP ON ERROR IF SO
U 1D29, CD82.35 :9643      NOLOOP.TA.7:
                                JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 8
U 1D2A, 09D1.89 :9644      LOOP.TA.8:
                                JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC
                                XOR WR[3] WITH LS[ZERO] TO Q, ;READ 0'S
U 1D2B, 8870.5C :9645      DT(LONG)&SET.ALU.CC
U 1D2C, 8870.9C :9646      MOV WR[3] TO WR[0]      ;SET UP RECEIVED DATA IN WRO
U 1D2D, CD9D.B5 :9647      CLR WR[1]      ;SET UP EXPECTED DATA IN WR1
U 1D2E, A006.15 :9648      JMP.IF[NEQ] TO [TA.8.ERR] ;REPORT ERROR IF WRO NOT CLEAR
U 1D2F, 2F82.95 :9649      MOV LS[ONES] TO WR[3] ;WRITE 1'S
U 1D30, 09D3.71 :9650      XOR WR[3] WITH LS[ONES] TO Q, ;READ 1'S
U 1D31, 869F.95 :9651      DT(LONG)&SET.ALU.CC
                                SKIP.IF[NEQ]      ;SKIP NEXT INSTRUCTION IF ALL BITS NOT SET
U 1D32, 4D9F.B5 :9652      JMP [TA.9]      ;GO TO NEXT SECTION
U 1D33, DB00.01 :9653      MOV WR[3] TO WR[0]      ;SET UP RECEIVED DATA IN WRO
U 1D34, 89D3.C4 :9654      MOV LS[ONES] TO WR[1] ;SET UP EXPECTED DATA IN WR1
U 1D35, A006.15 :9655      TA.8.ERR:
                                JSR [MOV.EX.RE]
                                MISC [SET.CP.ATTN] ;REPORT ERROR 8 TO CONSOLE
U 1D36, 369E.95 :9656      WAIT.TA.8:
                                JMP [WAIT.TA.8] ;WAIT FOR CONSOLE TO RESPOND
U 1D37, 8870.0C :9657      JMP [LOOP.TA.8] ;LOOP ON ERROR IF ENABLED
U 1D38, 10E0.15 :9658      JMP [NOLOOP.TA.8] ;GO TO NEXT SECTION IF NOT
U 1D39, 89D3.94 :9659      TA.9:
                                MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
U 1D3A, 09D2.C4 :9660
U 1D3B, 89D3.F4 :9661
U 1D3C, 36A0.15 :9662
  
```


ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST A - Working Re 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST A - Working Registers Test (M8390 CPU Module)

Fiche 2 Frame J1

Sequence 215
Rev 01.00 Page 209

```
:9678      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,      ;CURRENT ERROR?
U 1D3D, CD82,35 :9679      DT(LONG)&SET.ALU.CC
U 1D3E, 89D2,C9 :9680      JMP.IF[EQL] TO [LOOP.TA.8]      ;LOOP ON ERROR IF SO
:9681      NOLOOP.TA.8:
U 1D3F, 8870,5C :9682      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 9
U 1D40, B69E,15 :9683      MOV LS[ONES] TO WR[0]      ;RESET ONES IN WRO
U 1D41, 369E,95 :9684      MOV LS[ONES] TO WR[1]      ;RESET ONES IN WR1
:9685      LOOP.TA.9:
U 1D42, 8870,9C :9686      JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC
:9687      XOR WR[3] WITH LS[ONES] TO Q,      ;READ 1'S
U 1D43, 4D9F,B5 :9688      DT(LONG)&SET.ALU.CC
U 1D44, A006,15 :9689      MOV WR[3] TO WR[0]      ;SET UP RECEIVED DATA IN WRO
U 1D45, 2F82,95 :9690      CLR WR[1]      ;SET UP EXPECTED DATA IN WR1
U 1D46, 89D4,D1 :9691      JMP.IF[NEQ] TO [TA.9.ERR]      ;REPORT ERROR IF WRO NOT CLEAR
U 1D47, 369D,95 :9692      MOV LS[ZERO] TO WR[3]      ;WRITE 0'S
:9693      XOR WR[3] WITH LS[ZERO] TO Q,      ;READ 0'S
U 1D48, CD9D,B5 :9694      DT(LONG)&SET.ALU.CC
U 1D49, DB00,01 :9695      SKIP.IF[NEQ]      ;SKIP NEXT INSTRUCTION IF ALL BITS NOT SET
U 1D4A, 09D5,24 :9696      JMP [TA.A]      ;GO TO NEXT SECTION
U 1D4B, A006,15 :9697      MOV WR[3] TO WR[0]      ;SET UP RECEIVED DATA IN WRO
U 1D4C, B69C,95 :9698      MOV LS[ZERO] TO WR[1]      ;SET UP EXPECTED DATA IN WR1
:9699      TA.9.ERR:
U 1D4D, 8870,0C :9700      JSR [MOV.EX.RE]
U 1D4E, 10E0,15 :9701      MISC [SET.CP.ATTN]      ;REPORT ERROR 9 TO CONSOLE
:9702      WAIT.TA.9:
U 1D4F, 09D4,F4 :9703      JMP [WAIT.TA.9]      ;WAIT FOR CONSOLE TO RESPOND
U 1D50, 89D4,24 :9704      JMP [LOOP.TA.9]      ;LOOP ON ERROR IF ENABLED
U 1D51, 89D5,54 :9705      JMP [NOLOOP.TA.9]      ;GO TO NEXT SECTION IF NOT
:9706      TA.A:
U 1D52, 36A0,15 :9707      MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9708      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1D53, CD82,35 :9709      DT(LONG)&SET.ALU.CC
U 1D54, 09D4,29 :9710      JMP.IF[EQL] TO [LOOP.TA.9]      ;LOOP ON ERROR IF SO
:9711      NOLOOP.TA.9:
U 1D55, 8870,5C :9712      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 10
:9713      LOOP.TA.A:
U 1D56, 8870,9C :9714      JSR [ISSUE.SS]      ;ISSUE SCOPE SYNC
:9715      XOR WR[2] WITH LS[ONES] TO Q,      ;READ 1'S
U 1D57, CD9F,35 :9716      DT(LONG)&SET.ALU.CC
U 1D58, 2004,15 :9717      MOV WR[2] TO WR[0]      ;SET UP RECEIVED DATA IN WRO
U 1D59, 2F82,95 :9718      CLR WR[1]      ;SET UP EXPECTED DATA IN WR1
U 1D5A, 09D6,11 :9719      JMP.IF[NEQ] TO [TA.A.ERR]      ;REPORT ERROR IF WRO NOT CLEAR
U 1D5B, B69D,15 :9720      MOV LS[ZERO] TO WR[2]      ;WRITE 0'S
:9721      XOR WR[2] WITH LS[ZERO] TO Q,      ;READ 0'S
U 1D5C, 4D9D,35 :9722      DT(LONG)&SET.ALU.CC
U 1D5D, DB00,01 :9723      SKIP.IF[NEQ]      ;SKIP NEXT INSTRUCTION IF ALL BITS NOT SET
U 1D5E, 89D6,64 :9724      JMP [TA.A]      ;GO TO NEXT SECTION
U 1D5F, 2004,15 :9725      MOV WR[2] TO WR[0]      ;SET UP RECEIVED DATA IN WRO
U 1D60, B69C,95 :9726      MOV LS[ZERO] TO WR[1]      ;SET UP EXPECTED DATA IN WR1
:9727      TA.A.ERR:
U 1D61, 8870,0C :9728      JSR [MOV.EX.RE]
U 1D62, 10E0,15 :9729      MISC [SET.CP.ATTN]      ;REPORT ERROR A TO CONSOLE
:9730      WAIT.TA.A:
U 1D63, 89D6,34 :9731      JMP [WAIT.TA.A]      ;WAIT FOR CONSOLE TO RESPOND
U 1D64, 89D5,64 :9732      JMP [LOOP.TA.A]      ;LOOP ON ERROR IF ENABLED
```



```

U 1D65, 89D6,94 :9733      JMP [NOLOOP.TA.A]          ;GO TO NEXT SECTION IF NOT
:9734
U 1D66, 36A0,15 :9735      TA.B:      MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9736      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
:9737      DT(LONG)&SET.ALU.CC
U 1D67, CD82,35 :9738      JMP.IF[EQ] TO [LOOP.TA.A] ;LOOP ON ERROR IF SO
U 1D68, 09D5,69 :9739
:9740      NOLOOP.TA.A:      JSR [INC.EN] ;INCREMENT ERROR NUMBER TO B
:9741      MOV LS[ONES] TO WR[1] ;RESTORE WR1
:9742
U 1D6B, 8870,9C :9743      LOOP.TA.B:      JSR [ISSUE.SS] ;ISSUE SCOPE SYNC
:9744      XOR WR[1] WITH LS[ONES] TO Q, ;READ 1'S
:9745      DT(LONG)&SET.ALU.CC
U 1D6C, CD9E,B5 :9746      MOV WR[1] TO WR[0] ;SET UP RECEIVED DATA IN WR0
U 1D6D, 2002,15 :9747      CLR WR[1] ;SET UP EXPECTED DATA IN WR1
U 1D6E, 2F82,95 :9748      JMP.IF[NEQ] TO [TA.B.ERR] ;REPORT ERROR IF WR0 NOT CLEAR
U 1D6F, 09D7,61 :9749      MOV LS[ZERO] TO WR[1] ;WRITE 0'S
U 1D70, B69C,95 :9750      XOR WR[1] WITH LS[ZERO] TO Q, ;READ 0'S
:9751      DT(LONG)&SET.ALU.CC
U 1D71, 4D9C,B5 :9752      SKIP.IF[NEQ] ;SKIP NEXT INSTRUCTION IF ALL BITS NOT SET
U 1D72, DB00,01 :9753      JMP [TA.C] ;GO TO NEXT SECTION
U 1D73, 89D7,B4 :9754      MOV WR[1] TO WR[0] ;SET UP RECEIVED DATA IN WR0
U 1D74, 2002,15 :9755      MOV LS[ZERO] TO WR[1] ;SET UP EXPECTED DATA IN WR1
:9756
U 1D76, 8870,0C :9757      TA.B.ERR:      JSR [MOV.EX.RE]
U 1D77, 10E0,15 :9758      MISC [SET.CP.ATTN] ;REPORT ERROR B TO CONSOLE
:9759
U 1D78, 89D7,84 :9760      WAIT.TA.B:      JMP [WAIT.TA.B] ;WAIT FOR CONSOLE TO RESPOND
U 1D79, 09D6,B4 :9761      JMP [LOOP.TA.B] ;LOOP ON ERROR IF ENABLED
U 1D7A, 89D7,E4 :9762      JMP [NOLOOP.TA.B] ;GO TO NEXT SECTION IF NOT
:9763
U 1D7B, 36A0,15 :9764      TA.C:      MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9765      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
:9766      DT(LONG)&SET.ALU.CC
U 1D7C, CD82,35 :9767      JMP.IF[EQ] TO [LOOP.TA.B] ;LOOP ON ERROR IF SO
U 1D7D, 89D6,B9 :9768
:9769      NOLOOP.TA.B:      JSR [INC.EN] ;INCREMENT ERROR NUMBER TO C
U 1D7E, 8870,5C :9770      MOV LS[ONES] TO WR[0] ;RESTORE WR0
:9771
U 1D80, 8870,9C :9772      LOOP.TA.C:      JSR [ISSUE.SS] ;ISSUE SCOPE SYNC
:9773      XOR WR[0] WITH LS[ONES] TO Q, ;READ 1'S
:9774      DT(LONG)&SET.ALU.CC
U 1D81, 4D9E,35 :9775      CLR WR[1] ;SET UP EXPECTED DATA IN WR1
U 1D82, 2F82,95 :9776      JMP.IF[NEQ] TO [TA.C.ERR] ;REPORT ERROR IF WR0 NOT CLEAR
U 1D83, 09D8,91 :9777      MOV LS[ZERO] TO WR[0] ;WRITE 0'S
U 1D84, 369C,15 :9778      XOR WR[0] WITH LS[ZERO] TO Q, ;READ 0'S
:9779      DT(LONG)&SET.ALU.CC
U 1D85, CD9C,35 :9780      SKIP.IF[NEQ] ;SKIP NEXT INSTRUCTION IF ALL BITS NOT SET
U 1D86, DB00,01 :9781      JMP [TA.CA] ;GO TO NEXT SECTION
U 1D87, 89D8,E4 :9782      MOV LS[ZERO] TO WR[1] ;SET UP EXPECTED DATA IN WR1
:9783
U 1D89, 8870,0C :9784      TA.C.ERR:      JSR [MOV.EX.RE]
U 1D8A, 10E0,15 :9785      MISC [SET.CP.ATTN] ;REPORT ERROR C TO CONSOLE
:9786
U 1D8B, 89D8,B4 :9787      WAIT.TA.C:      JMP [WAIT.TA.C] ;WAIT FOR CONSOLE TO RESPOND
    
```

```

ZZ-ENKCA-1.0      : ENKCA.MIC      TEST A - Working Re 3-MAR-82      Fiche 2      Frame L1      Sequence 217
: ENKCA.MCR      MICRO2 1L(03)      3-MAR-82 09:34:35      ENKCA Micro-diagnostic for DAP module      Rev 01.00      Page 211
: ENKCA.MIC      TEST A - Working Registers Test (M8390 CPU Module)

U 1D8C, 09D8,04 :9788      JMP [LOOP.TA.C]      ;LOOP ON ERROR IF ENABLED
U 1D8D, 09D9,14 :9789      JMP [END.TA]      ;GO TO END OF TEST IF NOT
:9790      TA.CA:
U 1D8E, 36A0,15 :9791      MOV LS[PREVIOUS.ERROR] TO WR[0] ;GET PREVIOUS ERROR NUMBER
:9792      XOR WR[0] WITH LS[ERROR.NUMBER] TO Q, ;CURRENT ERROR?
U 1D8F, CD82,35 :9793      DT(LONG)&SET.ALU.CC
U 1D90, 89D8,09 :9794      JMP.IF[EQL] TO [LOOP.TA.C] ;LOOP ON ERROR IF SO
:9795      END.TA:

```


Page "TEST B - 2901 R PLUS S Test (M8390 CPU Module)"

Test Description:

This test checks the R PLUS S function of the 2901. The instruction used is ADD LS TO WR and the 2901 control lines are as follows: SRC=D.A FUNC=R PLUS S DST=WRITE.B.F CIN=NO CIN. To check the carry in to bit 0 the instruction ADD LS TO WR + 1 is used. The control lines for this instruction is the same as above except CIN=CIN to force a carry into bit 0. The data path is as follows: WR 0 is written from LS with a test pattern. Then another data pattern from LS is added to WR 0. WR 1 is loaded with the expected result and the error routine called to check for a failure. The data patterns used are as follows: all 1's and 0's, 0's and all 1's, 0's and 0's, all 1's and all 1's, and shifting 1's and shifting 1's. These patterns will check all possible combinations of patterns to the 2901 ALU, including a carry into each pattern and the GEN and PROP logic within the 2901 and as outputs. The instruction ADD LS TO WR + 1, which checks the carry in to bit 0, is used with the first four patterns only. This is sufficient to check all patterns on bit 0.

Assumptions:

It is assumed that the DST, and SRC of the 2901 for this function are working correctly as verified by previous tests.

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load WR 0 with all 1's and execute ADD LS TO WR with LS containing 0's. Load WR 1 with expected result (should be all 1's) and check.
- 3) Repeat step 2 with LS containing 1's and WR containing 0's. Result should be all 1's.
- 4) Repeat step 2 with WR 0 containing 0's. Result should be 0.
- 5) Repeat step 2 with WR 0 and LS containing all 1's. Result should be FFFFFFFE.
- 6) Load WR 0 with 0 and execute ADD LS TO WR + 1 with LS containing 0. Load WR 1 with expected result (should be 1) and check.
- 7) Repeat step 6 with LS containing 1's. Result should be 0.
- 8) Repeat step 6 with WR 0 containing 1's. Result should be 0.
- 9) Repeat step 6 with WR 0 and LS containing all 1's. Result should be all 1's.
- 10) Set up WR 0 with a shifting 1 pattern and execute ADD LS TO WR with LS containing a shifting 1 pattern. Load WR 1 with expected result and check.
- 11) Shift both data patterns left one bit and repeat step 11. Repeat this step until all 32 bits have been tested

Errors:

- Error 1 - ADD LS TO WR with data of 0 in LS and 1's in the WR failed.
Error 2 - ADD LS TO WR with data of 1's in LS and 0 in the WR failed.

:9851 Error 3 - ADD LS TO WR with data of 0 in LS and 0 in the WR failed.
:9852 Error 4 - ADD LS TO WR with data of 1's in LS and 1's in the WR failed.
:9853 Error 5 - ADD LS TO WR + 1 with data of 0 in LS and 0 in the WR failed.
:9854 Error 6 - ADD LS TO WR + 1 with data of 1's in LS and 0 in the WR fail-
:9855 ed.
:9856 Error 7 - ADD LS TO WR + 1 with data of 0's in LS and 1's in the WR
:9857 failed.
:9858 Error 8 - ADD LS TO WR + 1 with data of 1's in LS and 1's in the WR
:9859 failed.
:9860 Error 9 - ADD LS TO WR with data of shifting 1's in LS and shifting
:9861 1's in the WR failed.
:9862
:9863
:9864

Debug:

:9865 Error 1 - This error indicates a failure in the 2901 or it's control
:9866 lines. Refer to the 2901 control line description at the beginning
:9867 of this listing and the description of this test and check the control
:9868 lines at the 2901. Also check that the carry in to pin 29 of the low
:9869 2901 is 0. If the control lines or carry in are in error, check the
:9870 DEST CTL ROM and SRC.ALU CTL PAL outputs. The input to these IC's
:9871 for CSR 22 through CSR 14 should be 1 0000 0001(B) respectively.
:9872 Another possibility is that the CARRY SKIPPER (carry look-ahead) which
:9873 feeds the carry in to pin 29 of each 2901 is failing. The carry in
:9874 to each 2901 should be low and the signal ALU CARRY-IN H to the carry
:9875 skipper should be low. If 4 bits or less are failing, suspect a
:9876 bad 2901.
:9877

:9878 Error 2 - If this is the first error, it most likely indicates a
:9879 problem with a 2901 since the control lines are identical to error 1.
:9880

:9881 Error 3 - See error 2.
:9882

:9883 Error 4 - This error indicates a problem with a 2901 or carry logic.
:9884 It is the first pattern which checks a carry by asserting GEN. The
:9885 control lines are identical to those described in error 1 above but if
:9886 this is the first error, the control lines are probably OK. The carry
:9887 in pin (pin 29) of each 2901 (except the low order 2901) should be
:9888 asserted during the ADD function. This carry comes from the carry
:9889 skipper logic driven by the PROP and GEN outputs of the 2901. Each
:9890 2901 should be generating a low on its GEN output (pin 32) with the
:9891 data used in this test. The GEN going into the carry skipper is all
:9892 that is necessary to produce the CX OUT for the next 2901. If no GEN
:9893 is coming out of the 2901, the 2901 is bad. If GEN is going into the
:9894 skipper but no carry is going out, the skipper is bad. The internal
:9895 carry in the 2901 could be malfunctioning, which would cause an error
:9896 in the data even though GEN and carry in are OK.
:9897

:9898 Error 5 - This error indicates a failure in the low 2901, the control
:9899 lines or carry in (ALU CARRY-IN H), or the CARRY SKIPPER (look-ahead).
:9900 Also check that ALU CARRY-IN H at the low order CARRY SKIPPER is high
:9901 (asserted). If the control lines or ALU CARRY-IN is in error, check
:9902 the outputs at the DEST CTL ROM and SRC.ALU CTL PAL. The inputs to
:9903 IC's for CSR 22 through CSR 14 should be 1 0001 1001(B) respectively.
:9904 If the control lines are OK check that pin 29 of the low 2901 is
:9905 high. If not suspect the CARRY SKIPPER. Otherwise suspect the 2901.

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) TEST B - 2901 R PLU 3-MAR-82
TEST B - 2901 R PLUS S Test (M8390 CPU Module)

Fiche 2 Frame B2

Sequence 220 Page 214
Rev 01.00

:9906
:9907
:9908
:9909
:9910
:9911
:9912
:9913
:9914
:9915
:9916
:9917
:9918
:9919
:9920
:9921
:9922
:9923
:9924
:9925
:9926
:9927
:9928
:9929
:9930
:9931
:9932
:9933
:9934
:9935
:9936
:9937
:9938
:9939
:9940
:9941
:9942
:9943
:9944
:9945
:9946
:9947
:9948
:9949
:9950
:9951
:9952
:9953
:9954
:9955
:9956
:9957
:9958
:9959
:9960

Error 6 - If this is the first error, suspect a bad 2901 or a problem with the carry skipper. Control lines are identical to that in error 5 and are not likely to be causing the problem. The carry skipper is responsible for asserting the carry in to each 2901. Check that pin 29 is asserted during the ADD instruction. If not it is possible that the PROP output of the 2901 is in error. PROP should be asserted (low) on all 2901's during the add instruction. The carry skipper should be asserting CX OUT for each bit due to PROP being asserted and ALU CARRY IN being asserted. The signal LOW WORD CARRY H is necessary to produce the carry in to the 2901 producing bit 16 of the result. This comes out of the 74S51 fed by the CARRY SKIPPER. If the carry in is OK at the 2901 that is producing the bit in error, suspect the 2901 itself.

Error 7 - If this is the first error, the 2901 is most suspect. Check the PROP output as described in error 6 if the carry in on pin 29 is not asserted to the 2901 producing the bit in error. If carry in is OK, suspect the 2901 itself.

Error 8 - This error is similar to error 7 above. The control lines are identical to that described in error 5. The most likely problem if this is the first error is in the low order 2901. The carry in to this chip should be high producing all ones at the output.

Error 9 - If this is the first error, the control lines out of the DEST CTL ROM and SRC.ALU CTL PAL should be checked first. The output is the same as in error 1, but the inputs to these IC's from CSR 22 through CSR 14 should be 1 0000 0011(B) respectively. If the control lines are OK suspect a bad 2901 as the internal carry logic may be failing.

T.B:--

U 1D91, B65E,15
U 1D92, 3E80,15
U 1D93, 10E0,15
U 1D94, 09D9,44
U 1D95, 2F80,15
U 1D96, 3E8A,15
U 1D97, BEA0,15
U 1D98, 2040,15
U 1D99, BE82,15
U 1D9A, 2040,15
U 1D9B, 3E8C,15

MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN] ;BIT15 INDICATES START OF TEST TO CONSOLE
ISSUE CPU ATTN TO CONSOLE
WAIT.TB.0:
JMP [WAIT.TB.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR WR[0] ;SET WRO TO 0
MOV WR[0] TO LS[ERROR.MASK] ;CLEAR ERROR MASK.
MOV WR[0] TO LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
INC WR[0] ;SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
INC WR[0] ;SET WRO TO 2
MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE

LOOP.TB.1:

U 1D9C, B69E,15
U 1D9D, C09C,15
U 1D9E, 369E,95
U 1D9F, 0869,3C
U 1DA0, 89D9,C4
U 1DA1, 8870,5C

MOV LS[ONES] TO WR[0] ;SET ALL BITS IN WRO
ADD LS[ZERO] TO WR[0] ;TRY ADD INS. WITH 0 FROM LS
MOV LS[ONES] TO WR[1] ;SET UP EXPECTED RESULT IN WR1
JSR [CHECK.RESULT] ;CHECK RESULT
JMP [LOOP.TB.1] ;LOOP ON ERROR IF ENABLED
JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) TEST B - 2901 R PLU 3-MAR-82
TEST B - 2901 R PLUS S Test (M8390 CPU Module)

Fiche 2 Frame C2

Sequence 221
Rev 01.00

Page 215

```
:9961 LOOP.TB.2:
U 1DA2, 2F80,15 :9962 CLR WR[0] ;CLEAR ALL BITS IN WRO
U 1DA3, 409E,15 :9963 ADD LS[ONES] TO WR[0] ;TRY ADD INS. WITH #FFFFFFF FROM LS
U 1DA4, 369E,95 :9964 MOV LS[ONES] TO WR[1] ;PUT EXPECTED RESULT IN WR1
U 1DA5, 0869,3C :9965 JSR [CHECK.RESULT] ;CHECK RESULT
U 1DA6, 09DA,24 :9966 JMP [LOOP.TB.2] ;LOOP ON ERROR IF ENABLED
U 1DA7, 8870,5C :9967 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 3
:9968 LOOP.TB.3:
U 1DA8, 2F80,15 :9969 CLR WR[0] ;SET WRO TO 0
U 1DA9, C09C,15 :9970 ADD LS[ZERO] TO WR[0] ;TRY ADD INS. WITH 0 FROM LS
U 1DAA, 2F82,95 :9971 CLR WR[1] ;PUT EXPECTED RESULT IN WR1
U 1DAB, 0869,3C :9972 JSR [CHECK.RESULT] ;CHECK RESULT
U 1DAC, 09DA,84 :9973 JMP [LOOP.TB.3] ;LOOP ON ERROR IF ENABLED
U 1DAD, 8870,5C :9974 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 4
:9975 LOOP.TB.4:
U 1DAE, B69E,15 :9976 MOV LS[ONES] TO WR[0] ;SET ALL BITS IN WRO
U 1DAF, 409E,15 :9977 ADD LS[ONES] TO WR[0] ;TRY ADD INS WITH #FFFFFFF FROM LS
U 1DB0, 369E,95 :9978 MOV LS[ONES] TO WR[1] ;PUT EXPECTED DATA
U 1DB1, C540,95 :9979 BIC LS[BIT0] TO WR[1] ; IN WR1
U 1DB2, 0869,3C :9980 JSR [CHECK.RESULT] ;CHECK RESULT
U 1DB3, 09DA,E4 :9981 JMP [LOOP.TB.4] ;LOOP ON ERROR IF ENABLED
U 1DB4, 8870,5C :9982 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 5
:9983 LOOP.TB.5:
U 1DB5, 2F80,15 :9984 CLR WR[0] ;CLEAR ALL BITS IN WRO
U 1DB6, C69C,15 :9985 ADD (LS[ZERO] TO WR[0])+1 ;TRY ADD+1 INS. WITH 0 FROM LS
U 1DB7, 2F82,95 :9986 CLR WR[1] ;CLEAR WR1
U 1DB8, 2042,95 :9987 INC WR[1] ;SET WR1 TO 1 FOR EXPECTED RESULT
U 1DB9, 0869,3C :9988 JSR [CHECK.RESULT] ;CHECK RESULT
U 1DBA, 09DB,54 :9989 JMP [LOOP.TB.5] ;LOOP ON ERROR IF ENABLED
U 1DBB, 8870,5C :9990 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 6
:9991 LOOP.TB.6:
U 1DBC, 2F80,15 :9992 CLR WR[0] ;CLEAR WRO
U 1DBD, 469E,15 :9993 ADD (LS[ONES] TO WR[0])+1 ;TRY ADD+1 INS. WITH #FFFFFFF FROM LS
U 1DBE, 2F82,95 :9994 CLR WR[1] ;PUT EXPECTED RESULT IN WR1
U 1DBF, 0869,3C :9995 JSR [CHECK.RESULT] ;CHECK RESULT
U 1DC0, 09DB,C4 :9996 JMP [LOOP.TB.6] ;LOOP ON ERROR IF ENABLED
U 1DC1, 8870,5C :9997 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 7
:9998 LOOP.TB.7:
U 1DC2, B69E,15 :9999 MOV LS[ONES] TO WR[0] ;SET ALL BITS IN WRO
U 1DC3, C69C,15 :10000 ADD (LS[ZERO] TO WR[0])+1 ;TRY ADD+1 INS. WITH 0 FROM LS
U 1DC4, 2F82,95 :10001 CLR WR[1] ;PUT EXPECTED RESULT IN WR1
U 1DC5, 0869,3C :10002 JSR [CHECK.RESULT] ;CHECK RESULT
U 1DC6, 09DC,24 :10003 JMP [LOOP.TB.7] ;LOOP ON ERROR IF ENABLED
U 1DC7, 8870,5C :10004 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 8
:10005 LOOP.TB.8:
U 1DC8, B69E,15 :10006 MOV LS[ONES] TO WR[0] ;SET ALL BITS IN WRO
U 1DC9, 469E,15 :10007 ADD (LS[ONES] TO WR[0])+1 ;TRY ADD INS. WITH 1'S FROM LS
U 1DCA, 369E,95 :10008 MOV LS[ONES] TO WR[1] ;PUT EXPECTED DATA IN WR1
U 1DCB, 0869,3C :10009 JSR [CHECK.RESULT] ;CHECK RESULT
U 1DCC, 09DC,84 :10010 JMP [LOOP.TB.8] ;LOOP ON ERROR IF ENABLED
U 1DCD, 8870,5C :10011 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 9
U 1DCE, E5F8,15 :10012 CLR LS[OS] ;CLEAR THE OS REGISTER
:10013 LOOP.TB.9:
U 1DCF, 36F6,15 :10014 MOV LS[SHIFT.OS(4-0)] TO WR[0] ;PUT DATA IN WRO
U 1DD0, C0F6,15 :10015 ADD LS[SHIFT.OS(4-0)] TO WR[0] ;TRY ADD INS. USING A SHIFTING ONE PATTERN FROM LS
```



```

U 1DD1, B6F6,95 :10016      MOV LS[SHIFT.OS(4-0)] TO WR[1] ;PUT EXPECTED DATA
U 1DD2, A3D0,95 :10017      ASHL WR[1] ;IN WR1
U 1DD3, 0869,3C :10018      JSR [CHECK.RESULT] ;CHECK RESULT
U 1DD4, 89DC,F4 :10019      JMP [LOOP.TB.9] ;LOOP ON ERROR IF ENABLED
U 1DD5, 36F9,15 :10020      MOV LS[OS] TO WR[2] ;GET INDEX
U 1DD6, 2045,15 :10021      INC WR[2] ;INCREMENT IT
U 1DD7, C52D,15 :10022      BIC LS[#FFFFFF00] TO WR[2] ;CLEAR HIGH BITS
U 1DD8, BEF9,15 :10023      MOV WR[2] TO LS[OS] ;PUT IT BACK
:10024      XOR WR[2] WITH LS[#20] TO Q, ;IS IT EQUAL TO 32?
U 1DD9, CD4B,35 :10025      DT(LONG)&SET.ALU.CC
U 1DDA, 5B00,09 :10026      SKIP.IF[EQ] ;SKIP NEXT INSTRUCTION IF SO
U 1ddb, 89DC,F4 :10027      JMP [LOOP.TB.9] ;GO DO NEXT BIT
:10028      END.TB:

```

Page "TEST C - 2901 S MINUS R Test (M8390 CPU Module)"

Test Description:

This test checks the S MINUS R function of the 2901. The instruction used is SUB LS FROM WR - 1 and the 2901 control lines are as follows: SRC=D.A FUNC=S MINUS R DST=WRITE.B.F CIN=NO CIN. To check the carry in to bit 0 the instruction SUB LS FROM WR is used. The control lines for this instruction is the same as above except CIN=CIN to force a carry into bit 0. The data path is as follows: WR 0 is written from LS with a test pattern. Then another data pattern from LS is subtracted from WR 0. WR 1 is loaded with the expected result and the error routine called to check for a failure. The data patterns used are as follows: 0's and 0's, all 1's and 0's, 0's and all 1's, all 1's and all 1's, and shifting 1's and shifting 1's. These patterns will check all possible combinations of patterns to the 2901 ALU, including a carry into each pattern and the GEN and PROP logic within the 2901 and as outputs. The instruction SUB LS FROM WR, which checks the carry in to bit 0, is used with the first four patterns only. This is sufficient to check all patterns on bit 0.

Assumptions:

It is assumed that the DST, and SRC of the 2901 for this function are working correctly as verified by previous tests. It is also assumed that the CARRY SKIPPER (really a carry look-ahead) has been tested by the ADD test preceeding or is known to be good.

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load WR 0 with 0 and execute SUB LS FROM WR - 1 with LS containing 0. Load WR 1 with expected result (should be -1) and check.
- 3) Load WR 0 with 0 and execute SUB LS FROM WR with LS containing 0. Load WR 1 with expected result (should be 0) and check.
- 4) Repeat step 2 with LS containing 1's. Result should be 0.
- 5) Repeat step 3 with LS containing 1's. Result should be 1.
- 6) Repeat step 2 with WR 0 containing 1's. Result should be -2.
- 7) Repeat step 3 with WR 0 containing 1's. Result should be -1.
- 8) Repeat step 2 with WR 0 and LS containing all 1's. Result should be -1.
- 9) Repeat step 3 with WR 0 and LS containing all 1's. Result should be 0.
- 10) Set up WR 0 with a shifting 1 pattern and execute SUB LS FROM WR - 1 with LS containing a shifting 1 pattern. Load WR 1 with expected result (-1) and check.
- 11) Shift both data patterns left one bit and repeat step 11. Repeat this step until all 32 bits have been tested.

Errors:

- Error 1 - SUB LS FROM WR - 1 with data of 0 in LS and 0 in the WR failed.
Error 2 - SUB LS FROM WR with data of 0 in LS and 0 in the WR failed.

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST C - 2901 S MIN 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST C - 2901 S MINUS R Test (M8390 CPU Module)

F 2

Fiche 2 Frame F2

Sequence 224
Rev 01.00

Page 218

:10084 : Error 3 - SUB LS FROM WR - 1 with data of 1's in LS and 0 in the WR
:10085 : failed.
:10086 : Error 4 - SUB LS FROM WR with data of 1's in LS and 0 in the WR failed.
:10087 : Error 5 - SUB LS FROM WR - 1 with data of 0's in LS and 1's in the WR
:10088 : failed.
:10089 : Error 6 - SUB LS FROM WR with data of 0's in LS and 1's in the WR
:10090 : failed.
:10091 : Error 7 - SUB LS FROM WR - 1 with data of 1's in LS and 1's in the WR
:10092 : failed.
:10093 : Error 8 - SUB LS FROM WR with data of 1's in LS and 1's in the WR
:10094 : failed.
:10095 : Error 9 - SUB LS FROM WR - 1 with data of shifting 1's in LS and
:10096 : shifting 1's in the WR failed.
:10097 :
:10098 : Debug:
:10099 :
:10100 : Error 1 - This error indicates a failure in the 2901 or it's control
:10101 : lines. Refer to the 2901 control line description at the beginning
:10102 : of this listing and the description of this test and check the control
:10103 : lines at the 2901. Also check that the carry in to pin 29 of the low
:10104 : 2901 is 0. If the control lines or carry in are in error, check the
:10105 : DEST CTL ROM and SRC.ALU CTL PAL outputs. The input to these IC's
:10106 : for CSR 22 through CSR 14 should be 1 0001 0001(B) respectively.
:10107 : The signal ALU CARRY-IN H to the carry skipper should be low. If the
:10108 : control lines are OK, suspect a bad 2901.
:10109 :
:10110 : Error 2 - This error indicates a failure in the 2901, or the control
:10111 : lines. Also check that ALU CARRY-IN H at the low order CARRY SKIPPER
:10112 : is high (asserted). If the control lines or ALU CARRY-IN is in error,
:10113 : check the outputs at the DEST CTL ROM and SRC.ALU CTL PAL. The inputs
:10114 : to IC's for CSR 22 through CSR 14 should be 1 0000 0101(B) respectively.
:10115 : If the control lines are OK, then suspect the 2901.
:10116 :
:10117 : Error 3 - If this is the first error, it most likely indicates a
:10118 : problem with a 2901 since the control lines are identical to error 1.
:10119 :
:10120 : Error 4 - If this is the first error, then suspect a 2901 as the control
:10121 : lines are identical to error 2.
:10122 :
:10123 : Error 5 - See error 3.
:10124 :
:10125 : Error 6 - See error 4.
:10126 :
:10127 : Error 7 - See error 3.
:10128 :
:10129 : Error 8 - See error 4.
:10130 :
:10131 : Error 9 - If this is the first error, the control lines out of the
:10132 : DEST CTL ROM and SRC.ALU CTL PAL should be checked first. The output
:10133 : is the same as in error 1, but the inputs to these IC's from CSR 22
:10134 : through CSR 14 should be 1 0001 0011(B) respectively. If the control
:10135 : lines are OK suspect a bad 2901 as the internal carry logic may be
:10136 : failing.
:10137 :
:10138 : --

```

:10139 T.C:
U 1DDC, B65E,15 :10140 MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
U 1DDD, 3E80,15 :10141 MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
:10142 ;BIT15 INDICATES START OF TEST TO CONSOLE
U 1DDE, 10E0,15 :10143 MISC [SET.CP.ATTN] ;ISSUE CPU ATTN TO CONSOLE
:10144
WAIT.TC.0:
U 1DDF, 09DD,F4 :10145 JMP [WAIT.TC.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 1DE0, E58A,15 :10146 CLR LS[ERROR.MASK] ;CLEAR ERROR MASK.
U 1DE1, 65A0,15 :10147 CLR LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
U 1DE2, 2F80,15 :10148 CLR WR[0]
U 1DE3, 2040,15 :10149 INC WR[0] ;SET WRO TO 1
U 1DE4, BE82,15 :10150 MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
U 1DE5, 2040,15 :10151 INC WR[0] ;SET WRO TO 2
U 1DE6, 3E8C,15 :10152 MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
:10153
LOOP.TC.1:
U 1DE7, 2F80,15 :10154 CLR WR[0] ;SET WRO TO 0
U 1DE8, 449C,15 :10155 SUB (LS[ZERO] FROM WR[0])-1 ;TRY SUB-1 INS WITH ZERO FROM LS
U 1DE9, 369E,95 :10156 MOV LS[ONES] TO WR[1] ;PUT EXPECTED DATA IN WR1
U 1DEA, 0869,3C :10157 JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1DEB, 89DE,74 :10158 JMP [LOOP.TC.1] ;LOOP ON ERROR IF ENABLED
U 1DEC, 8870,5C :10159 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
:10160
LOOP.TC.2:
U 1DED, 2F80,15 :10161 CLR WR[0] ;SET WRO TO 0
U 1DEE, 419C,15 :10162 SUB LS[ZERO] FROM WR[0] ;TRY SUB INS WITH ZERO FROM LS
U 1DEF, 2F82,95 :10163 CLR WR[1] ;PUT EXPECTED DATA IN WR1
U 1DF0, 0869,3C :10164 JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1DF1, 89DE,D4 :10165 JMP [LOOP.TC.2] ;LOOP ON ERROR IF ENABLED
U 1DF2, 8870,5C :10166 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 3
:10167
LOOP.TC.3:
U 1DF3, 2180,15 :10168 CLR WR[0] ;SET WRO TO 0
U 1DF4, C49E,15 :10169 SUB (LS[ONES] FROM WR[0])-1 ;TRY SUB-1 INS WITH #FFFFFFF FROM LS
U 1DF5, 2F82,95 :10170 CLR WR[1] ;PUT EXPECTED DATA IN WR1
U 1DF6, 0869,3C :10171 JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1DF7, 89DF,34 :10172 JMP [LOOP.TC.3] ;LOOP ON ERROR IF ENABLED
U 1DF8, 8870,5C :10173 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 4
:10174
LOOP.TC.4:
U 1DF9, 2F80,15 :10175 CLR WR[0] ;SET WRO TO 0
U 1DFA, C19E,15 :10176 SUB LS[ONES] FROM WR[0] ;TRY SUB INS WITH #FFFFFFF FROM LS
U 1DFB, 2F82,95 :10177 CLR WR[1] ;PUT EXPECTED DATA (1)
U 1DFC, 2042,95 :10178 INC WR[1] ;IN WR1
U 1DFD, 0869,3C :10179 JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1DFE, 89DF,94 :10180 JMP [LOOP.TC.4] ;LOOP ON ERROR IF ENABLED
U 1DFF, 8870,5C :10181 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 5
:10182
LOOP.TC.5:
U 1E00, B69E,15 :10183 MOV LS[ONES] TO WR[0] ;SET WRO TO -1
U 1E01, 449C,15 :10184 SUB (LS[ZERO] FROM WR[0])-1 ;TRY SUB-1 INS WITH ZERO FROM LS
U 1E02, 369E,95 :10185 MOV LS[ONES] TO WR[1] ;PUT EXPECTED DATA (-2)
U 1E03, C540,95 :10186 BIC LS[BIT0] TO WR[1] ;IN WR1
U 1E04, 0869,3C :10187 JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1E05, 89E0,04 :10188 JMP [LOOP.TC.5] ;LOOP ON ERROR IF ENABLED
U 1E06, 8870,5C :10189 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 6
:10190
LOOP.TC.6:
U 1E07, B69E,15 :10191 MOV LS[ONES] TO WR[0] ;SET WRO TO -1
U 1E08, 419C,15 :10192 SUB LS[ZERO] FROM WR[0] ;TRY SUB INS WITH ZERO FROM LS
U 1E09, 369E,95 :10193 MOV LS[ONES] TO WR[1] ;PUT EXPECTED DATA (-1) IN WR1
    
```


ZZ-ENKCA-1.0 : ENKCA.MIC
 : ENKCA.MCR
 : ENKCA.MIC

MICRO2 1L(03) TEST C - 2901 S MIN 3-MAR-82
 TEST C - 2901 S MINUS R Test (M8390 CPU Module)

H 2
 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module Rev 01.00

Fiche 2 Frame H2
 Sequence 226 Page 220

```

U 1EOA, 0869,3C :10194      JSR [CHECK.RESULT]      ;CHECK THE RESULT
U 1EOB, 09E0,74 :10195      JMP [LOOP.TC.6]          ;LOOP ON ERROR IF ENABLED
U 1EOC, 8870,5C :10196      JSR [INC.EN]              ;INCREMENT ERROR NUMBER TO 7
                        :10197
LOOP.TC.7:
U 1EOD, B69E,15 :10198      MOV LS[ONES] TO WR[0]      ;SET WRO TO -1
U 1EOE, C49E,15 :10199      SUB (LS[ONES] FROM WR[0])-1 ;TRY SUB-1 INS WITH #FFFFFFF FROM LS
U 1EOF, 369E,95 :10200      MOV LS[ONES] TO WR[1]      ;PUT EXPECTED DATA (-1) IN WR1
U 1E10, 0869,3C :10201      JSR [CHECK.RESULT]      ;CHECK THE RESULT
U 1E11, 09E0,D4 :10202      JMP [LOOP.TC.7]          ;LOOP ON ERROR IF ENABLED
U 1E12, 8870,5C :10203      JSR [INC.EN]              ;INCREMENT ERROR NUMBER TO 8
                        :10204
LOOP.TC.8:
U 1E13, B69E,15 :10205      MOV LS[ONES] TO WR[0]      ;SET WRO TO -1
U 1E14, C19E,15 :10206      SUB LS[ONES] FROM WR[0]    ;TRY SUB INS WITH #FFFFFFF FROM LS
U 1E15, 2F82,95 :10207      CLR WR[1]                ;PUT EXPECTED DATA IN WR1
U 1E16, 0869,3C :10208      JSR [CHECK.RESULT]      ;CHECK THE RESULT
U 1E17, 09E1,34 :10209      JMP [LOOP.TC.8]          ;LOOP ON ERROR IF ENABLED
U 1E18, 8870,5C :10210      JSR [INC.EN]              ;INCREMENT ERROR NUMBER TO 9
U 1E19, E5F8,15 :10211      CLR LS[OS]                ;CLEAR THE OS REGISTER
                        :10212
LOOP.TC.9:
U 1E1A, 36F6,15 :10213      MOV LS[SHIFT.OS(4-0)] TO WR[0] ;SET SINGLE BIT IN WRO
U 1E1B, 44F6,15 :10214      SUB (LS[SHIFT.OS(4-0)] FROM WR[0])-1 ;TRY SUB INS WITH SHIFTING PATTERN FROM LS
U 1E1C, 369E,95 :10215      MOV LS[ONES] TO WR[1]      ;PUT EXPECTED RESULT (-1) IN WR1
U 1E1D, 0869,3C :10216      JSR [CHECK.RESULT]      ;CHECK THE RESULT
U 1E1E, 09E1,A4 :10217      JMP [LOOP.TC.9]          ;LOOP ON ERROR IF ENABLED
U 1E1F, 36F9,15 :10218      MOV LS[OS] TO WR[2]        ;GET THE INDEX
U 1E20, 2045,15 :10219      INC WR[2]                ;INCREMENT IT
U 1E21, C52D,15 :10220      BIC LS[FFFFFF00] TO WR[2]    ;CLEAR HIGH BITS
U 1E22, BEF9,15 :10221      MOV WR[2] TO LS[OS]        ;PUT IT BACK
                        :10222
                        XOR WR[2] WITH LS[#20] TO Q, ;IS IT EQUAL TO 32?
                        DT(LONG)&SET.ALU.CC
U 1E23, CD4B,35 :10223      SKIP.IF[EQL]          ;SKIP NEXT INS IF SO
U 1E24, 5B00,09 :10224      JMP [LOOP.TC.9]          ;GO DO NEXT BIT
U 1E25, 09E1,A4 :10225
                        :10226
END.TC:
  
```

Page "TEST D - 2901 R MINUS S Test (M8390 CPU Module)"

++

Test Description:

This test checks the R MINUS S function of the 2901. The instruction used is SUB (WR FROM LS) - 1 and the 2901 control lines are as follows: SRC=D.A FUNC=R MINUS S DST=NOP CIN=NO CIN. To check the carry in to bit 0 the instruction SUB WR FROM LS TO WR is used. The control lines for this instruction are as follows: SRC=D.A FUNC= R.MINUS.S DST=WRITE.B.F CIN=CIN. The data path is as follows: WR 0 is written from LS with a test pattern. Then a temporary location is written in LS with this data. WR 0 is again written with data, then SUB (WR FROM LS) - 1 is executed, modifying LS. The data now in LS is written into WR 0 as received data. WR 1 is loaded with the expected result and the error routine called to check for a failure. The data path for the SUB WR FROM LS TO WR instruction is as follows: WR 1 is written from LS with a test pattern. The instruction SUB WR FROM LS TO WR is executed, putting the result in WR 0. WR 1 is loaded with the expected result and the error routine called to check for a failure. The data patterns used are as follows: 0's and 0's, all 1's and 0's, 0's and all 1's, all 1's and all 1's, and shifting 1's and shifting 1's. These patterns will check all possible combinations of patterns to the 2901 ALU, including a carry into each pattern and the GEN and PROP logic within the 2901 and as outputs. The instruction SUB WR FROM LS TO WR, which checks the carry in to bit 0, is used with the first four patterns only. This is sufficient to check all patterns on bit 0.

Assumptions:

It is assumed that the DST, and SRC of the 2901 for this function are working correctly as verified by previous tests. It is also assumed that the CARRY SKIPPER (really a carry look-ahead) has been tested by the ADD test preceeding or is known to be good.

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load LS with 0 and execute SUB (WR FROM LS) - 1 with WR containing 0. Mov result from LS to WR 0. Load WR 1 with expected result (should be -1) and check.
- 3) Load LS with 0 and execute SUB WR FROM LS TO WR with WR containing 0. Load WR 1 with expected result (should be 0) and check.
- 4) Repeat step 2 with WR containing 1's. Result should be 0.
- 5) Repeat step 3 with WR containing 1's. Result should be 1.
- 6) Repeat step 2 with LS containing 1's. Result should be -2.
- 7) Repeat step 3 with LS containing 1's. Result should be -1.
- 8) Repeat step 2 with WR and LS containing all 1's. Result should be -1.
- 9) Repeat step 3 with WR and LS containing all 1's. Result should be 0.
- 10) Set up LS with a shifting 1 pattern and execute SUB (WR FROM LS) - 1 with WR containing a shifting 1 pattern. Load WR 1 with expected

:10282
:10283
:10284
:10285
:10286
:10287
:10288
:10289
:10290
:10291
:10292
:10293
:10294
:10295
:10296
:10297
:10298
:10299
:10300
:10301
:10302
:10303
:10304
:10305
:10306
:10307
:10308
:10309
:10310
:10311
:10312
:10313
:10314
:10315
:10316
:10317
:10318
:10319
:10320
:10321
:10322
:10323
:10324
:10325
:10326
:10327
:10328
:10329
:10330
:10331
:10332
:10333
:10334
:10335
:10336

result (-1) and check.

11) Shift both data patterns left one bit and repeat step 11. Repeat this step until all 32 bits have been tested

Errors:

- Error 1 - SUB (WR FROM LS) - 1 with data of 0 in WR and 0 in the LS failed.
- Error 2 - SUB WR FROM LS TO WR with data of 0 in WR and 0 in the LS failed.
- Error 3 - SUB (WR FROM LS) - 1 with data of 1's in WR and 0 in the LS failed.
- Error 4 - SUB WR FROM LS TO WR with data of 1's in WR and 0 in the LS failed.
- Error 5 - SUB (WR FROM LS) - 1 with data of 0's in WR and 1's in the LS failed.
- Error 6 - SUB WR FROM LS TO WR with data of 0's in WR and 1's in the LS failed.
- Error 7 - SUB (WR FROM LS) - 1 with data of 1's in WR and 1's in the LS failed.
- Error 8 - SUB WR FROM LS TO WR with data of 1's in WR and 1's in the LS failed.
- Error 9 - SUB (WR FROM LS) - 1 with data of shifting 1's in WR and shifting 1's in the LS failed.

Debug:

- Error 1 - This error indicates a failure in the 2901 or it's control lines. Refer to the 2901 control line description at the beginning of this listing and the description of this test and check the control lines at the 2901. Also check that the carry in to pin 29 of the low 2901 is 0. If the control lines or carry in are in error, check the DEST CTL ROM and SRC.ALU CTL PAL outputs. The input to these IC's for CSR 22 through CSR 14 should be 1 1010 0000(B) respectively. The signal ALU CARRY-IN H to the carry skipper should be low. If the control lines are OK, suspect a bad 2901.
- Error 2 - This error indicates a failure in the 2901, or the control lines. Also check that ALU CARRY-IN H at the low order CARRY SKIPPER is high (asserted). If the control lines or ALU CARRY-IN is in error, check the outputs at the DEST CTL ROM and SRC.ALU CTL PAL. The inputs to IC's for CSR 22 through CSR 14 should be 1 0111 0101(B) respectively. If the control lines are OK, then suspect the 2901.
- Error 3 - If this is the first error, it most likely indicates a problem with a 2901 since the control lines are identical to error 1.
- Error 4 - If this is the first error, then suspect a 2901 as the control lines are identical to error 2.
- Error 5 - See error 3.
- Error 6 - See error 4.
- Error 7 - See error 3.

```

:10337
:10338
:10339
:10340
:10341
:10342
:10343
:10344
:10345
:10346
:10347
:10348
:10349
:10350
:10351
:10352
:10353
:10354
:10355
:10356
:10357
:10358
:10359
:10360
:10361
:10362
:10363
:10364
:10365
:10366
:10367
:10368
:10369
:10370
:10371
:10372
:10373
:10374
:10375
:10376
:10377
:10378
:10379
:10380
:10381
:10382
:10383
:10384
:10385
:10386
:10387
:10388
:10389
:10390
:10391

Error 8 - See error 4.

Error 9 - If this is the first error, suspect a bad 2901 as the
internal carry logic may be failing. The control information is
identical to error 1.

T.D:
MOV LS[BEGIN.TEST] TO WR[0]      ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN]              ;BIT15 INDICATES START OF TEST TO CONSOLE
ISSUE CPU ATTN TO CONSOLE

WAIT.TD.0:
JMP [WAIT.TD.0]                ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR LS[ERROR.MASK]              ;CLEAR ERROR MASK.
CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
CLR WR[0]
INC WR[0]                       ;SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER]    ;SET ERROR NUMBER TO FIRST ERROR
INC WR[0]                       ;SET WRO TO 2
MOV WR[0] TO LS[MODULE.NUM]      ;SET UP MODULE CODE FOR CPU MODULE

LOOP.TD.1:
CLR WR[0]                       ;SET WRO TO 0
CLR LS[T8]                      ;SET LS TO 0
SUB (WR[0] FROM LS[T8])-1        ;TRY SUB-1 INS WITH ZERO FROM LS
MOV LS[T8] TO WR[0]              ;PUT RECEIVED DATA IN WRO
MOV LS[ONES] TO WR[1]            ;PUT EXPECTED RESULT (-1) IN WR1
JSR [CHECK.RESULT]               ;CHECK THE RESULT
JMP [LOOP.TD.1]                  ;LOOP ON ERROR IF ENABLED
JSR [INC.EN]                     ;INCREMENT ERROR NUMBER TO 2

LOOP.TD.2:
CLR WR[0]                       ;SET WRO TO 0
CLR LS[T8]                      ;SET LS TO 0
SUB WR[0] FROM LS[T8] TO WR      ;TRY SUB INS WITH ZERO FROM LS
CLR WR[1]                        ;PUT EXPECTED RESULT IN WR1
JSR [CHECK.RESULT]               ;CHECK THE RESULT
JMP [LOOP.TD.2]                  ;LOOP ON ERROR IF ENABLED
JSR [INC.EN]                     ;INCREMENT ERROR NUMBER TO 3

LOOP.TD.3:
MOV LS[ONES] TO WR[0]            ;SET WRO TO -1
CLR LS[T8]                      ;SET LS TO 0
SUB (WR[0] FROM LS[T8])-1        ;TRY SUB-1 INS WITH ZERO FROM LS
MOV LS[T8] TO WR[0]              ;PUT RECEIVED DATA IN WRO
CLR WR[1]                        ;PUT EXPECTED RESULT (0) IN WR1
JSR [CHECK.RESULT]               ;CHECK THE RESULT
JMP [LOOP.TD.3]                  ;LOOP ON ERROR IF ENABLED
JSR [INC.EN]                     ;INCREMENT ERROR NUMBER TO 4

LOOP.TD.4:
MOV LS[ONES] TO WR[0]            ;SET WRO TO -1
CLR LS[T8]                      ;SET LS TO 0
SUB WR[0] FROM LS[T8] TO WR      ;TRY SUB INS WITH ZERO FROM LS
CLR WR[1]                        ;PUT EXPECTED RESULT (1)
INC WR[1]                       ;IN WR1
JSR [CHECK.RESULT]               ;CHECK THE RESULT

```



```

ZZ-ENKCA-1.0      : ENKCA.MIC      L 2
: ENKCA.MCR        : ENKCA.MIC      3-MAR-82 09:34:33  FICHE 2 Frame L2  Sequence 230
: ENKCA.MIC        : ENKCA.MIC      3-MAR-82 09:34:33  ENKCA Micro-diagnostic for DAP module Rev 01.00  Page 224
MICRO2 1L(03)     : TEST D - 2901 R MINUS S Test (M8390 CPU Module)

U 1E4E, 89E4.84 :10392      JMP [LOOP.TD.4]      ;LOOP ON ERROR IF ENABLED
U 1E4F, 8870.5C :10393      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 5
:10394      LOOP.TD.5:      CLR WR[0]      ;SET WRO TO 0
U 1E50, 2F80.15 :10395      MOV LS[ONES] TO WR[2]
U 1E51, 369F.15 :10396      MOV WR[2] TO LS[T8]      ;SET LS TO -1
U 1E52, BE11.15 :10397      SUB (WR[0] FROM LS[T8])-1      ;TRY SUB-1 INS WITH #FFFFFFF FROM LS
U 1E53, 6810.15 :10398      MOV LS[T8] TO WR[0]      ;PUT RECEIVED DATA IN WRO
U 1E54, B610.15 :10399      MOV LS[ONES] TO WR[1]      ;PUT EXPECTED RESULT (-2)
U 1E55, 369E.95 :10400      BIC LS[BIT0] TO WR[1]      ; IN WR1
U 1E56, C540.95 :10401      JSR [CHECK.RESULT]      ;CHECK THE RESULT
U 1E57, 0869.3C :10402      JMP [LOOP.TD.5]      ;LOOP ON ERROR IF ENABLED
U 1E58, 89E5.04 :10403      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 6
U 1E59, 8870.5C :10404      LOOP.TD.6:      CLR WR[0]      ;SET WRO TO 0
:10405      MOV LS[ONES] TO WR[2]
U 1E5A, 2F80.15 :10406      MOV WR[2] TO LS[T8]      ;SET LS TO -1
U 1E5B, 369F.15 :10407      SUB WR[0] FROM LS[T8] TO WR      ;TRY SUB INS WITH #FFFFFFF FROM LS
U 1E5C, BE11.15 :10408      MOV LS[ONES] TO WR[1]      ;PUT EXPECTED RESULT (-1) IN WR1
U 1E5D, 5D10.15 :10409      JSR [CHECK.RESULT]      ;CHECK THE RESULT
U 1E5E, 369E.95 :10410      JMP [LOOP.TD.6]      ;LOOP ON ERROR IF ENABLED
U 1E5F, 0869.3C :10411      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 7
U 1E60, 89E5.A4 :10412      LOOP.TD.7:      MOV LS[ONES] TO WR[0]      ;SET WRO TO -1
:10413      MOV WR[0] TO LS[T8]      ;SET LS TO -1
U 1E61, 8870.5C :10414      SUB (WR[0] FROM LS[T8])-1      ;TRY SUB-1 INS WITH #FFFFFFF FROM LS
:10415      MOV LS[T8] TO WR[0]      ;PUT RECEIVED DATA IN WRO
U 1E62, B69E.15 :10416      MOV LS[ONES] TO WR[1]      ;PUT EXPECTED RESULT (-1) IN WR1
U 1E63, 3E10.15 :10417      JSR [CHECK.RESULT]      ;CHECK THE RESULT
U 1E64, 6810.15 :10418      JMP [LOOP.TD.7]      ;LOOP ON ERROR IF ENABLED
U 1E65, B610.15 :10419      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 8
U 1E66, 369E.95 :10420      LOOP.TD.8:      MOV LS[ONES] TO WR[0]      ;SET WRO TO -1
:10421      MOV WR[0] TO LS[T8]      ;SET LS TO -1
U 1E67, 0869.3C :10422      SUB WR[0] FROM LS[T8] TO WR      ;TRY SUB INS WITH #FFFFFFF FROM LS
:10423      CLR WR[1]      ;PUT EXPECTED RESULT IN WR1
U 1E68, 09E6.24 :10424      JSR [CHECK.RESULT]      ;CHECK THE RESULT
U 1E69, 8870.5C :10425      JMP [LOOP.TD.8]      ;LOOP ON ERROR IF ENABLED
:10426      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 9
U 1E6A, B69E.15 :10427      CLR LS[OS]      ;CLEAR THE OS REGISTER
U 1E6B, 3E10.15 :10428      LOOP.TD.9:      MOV LS[SHIFT.OS(4-0)] TO WR[0]      ;SET WRO TO SHIFTING 1 PATTERN
:10429      MOV WR[0] TO LS[T8]      ;SET LS TO SHIFTING 1 PATTERN
U 1E6C, 5D10.15 :10430      SUB (WR[0] FROM LS[T8])-1      ;TRY SUB-1 INS WITH SHIFTING 1 FROM LS
:10431      MOV LS[T8] TO WR[0]      ;PUT RECEIVED DATA IN WRO
U 1E6D, 2F82.95 :10432      MOV LS[ONES] TO WR[1]      ;PUT EXPECTED RESULT (-1) IN WR1
U 1E6E, 0869.3C :10433      JSR [CHECK.RESULT]      ;CHECK THE RESULT
U 1E6F, 89E6.A4 :10434      JMP [LOOP.TD.9]      ;LOOP ON ERROR IF ENABLED
:10435      MOV LS[OS] TO WR[2]      ;GET INDEX
U 1E70, 8870.5C :10436      INC WR[2]      ;INCREMENT IT
U 1E71, E5F8.15 :10437      BIC LS[FFFFFF00] TO WR[2]      ;CLEAR HIGH BITS
:10438      MOV WR[2] TO LS[OS]      ;PUT IT BACK
U 1E72, 36F6.15 :10439      XOR WR[2] WITH LS[#20] TO 0,      ;IS IT EQUAL TO 32?
:10440      DT(LONG)&SET.ALU.CC
U 1E73, 3E10.15 :10441      SKIP.IF[EQ]      ;SKIP NEXT INSTRUCTION IF SO
U 1E74, 6810.15 :10442
U 1E75, B610.15 :10443
U 1E76, 369E.95 :10444
U 1E77, 0869.3C :10445
U 1E78, 89E7.24 :10446
U 1E79, 36F9.15 :10447
U 1E7A, 2045.15 :10448
U 1E7B, C52D.15 :10449
U 1E7C, BEF9.15 :10450
U 1E7D, CD4B.35 :10451
U 1E7E, 5B00.09 :10452

```

ZZ-ENKCA-1.0 : ENKCA.MIC TEST D - 2901 R MIN 3-MAR-82 M 2
: ENKCA.MCR MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module Sequence 231
: ENKCA.MIC TEST D - 2901 R MINUS S Test (M8390 CPU Module) Rev 01.00 Page 225
U 1E7F, 89E7,24 :10447 JMP [LOOP.TD.9] :GO DO NEXT BIT
:10448 END.TD:

Page "TEST E - Q Register Test (M8390 CPU Module)"

Test Description:

This test checks the Q register of the 2901. The Q register will be loaded using an XOR WR WITH LS TO Q macro instruction. This instruction has been tested previously but only the condition codes were used. The value written into the Q reg was not checked. The Q register will be read using a MOV Q TO WR macro instruction. The control lines for the write (XOR) are as follows: SRC=D.A FUNC=R XOR S DST=LOAD.Q CIN=CIN. The control lines for the read (MOV) are as follows: SRC=0.Q FUNC=R OR S DST=WRITE.B.F CIN=CIN. The two new 2901 controls to be tested are DST=LOAD.Q on the write and SRC=0.Q on the read, in addition to the Q register itself. The data patterns written into the Q reg will be all 1's, all 0's, shifted 1's and shifted 0's. The data path is as follows: WR 1 is written from LS, then XOR WR WITH LS TO Q is executed writing the Q reg. The contents of the Q reg is written into WR 0, the expected result is already in WR 1, and the result is checked in the error routine.

Assumptions:

It is assumed that previous 2901 tests have run successfully. Please note that the error routine also uses XOR WR WITH LS TO Q and XOR WR WITH WR TO Q so the actual value of the Q reg will be modified during error checking. However this will be transparent to the user since the actual value printed in the error report comes from WR 0 where the contents of the Q reg have been moved.

Test Steps:

- 1) Set up the error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load WR 1 with all 0's and execute XOR WR[1] WITH LS TO Q using 0's from LS to preload the Q reg with 0's.
- 3) Load WR 1 with all 1's and execute XOR WR[1] WITH LS TO Q using zeros from LS.
- 4) Execute MOV Q TO WR[0], and check result for all 1's.
- 5) Repeat steps 2 and 3 using C's in WR 1 and checking for all 0's.
- 6) Repeat steps 2 and 3 using shifting 1's in WR 1 and checking for shifting 1's result.
- 7) Repeat steps 2 and 3 using shifting 0's in WR 1 and checking for shifting 0's result.

Errors:

- Error 1 - Q reg test failed with data of all ones.
Error 2 - Q reg test failed with data of all zeros.
Error 3 - Q reg test failed with data of shifting ones.
Error 4 - Q reg test failed with data of shifting zeros.

Debug:

Error 1 - This error indicates a problem in one of several areas. The

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST E - Q Register 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST E - Q Register Test (M8390 CPU Module)

Fiche 2 Frame B3

Sequence 233
Rev 01.00 Page 227

```
:10504 : Q register itself could be failing. This is a good possibility if all
:10505 : bits are not failing. The new source or destination modes in the 2901
:10506 : could be failing. This might show up as a group of 4 bits failing.
:10507 : Both of these failures means that the 2901 is faulty. Another possible
:10508 : problem may be in the control lines. See the 2901 control description
:10509 : in the beginning of this listing and the description of this test to
:10510 : determine the correct control line levels. These can be traced back
:10511 : to the DEST CTL ROM and ALU.SRC.CTL PAL. The inputs to these PALS
:10512 : for the write instruction (XOR) from CSR 22 through CSR 14 should be
:10513 : 1 0011 0101(B) respectively. For the read instruction (MOV) CSR 22
:10514 : through CSR 14 should be 0 1000 0110(B) respectively.
:10515 :
:10516 : Error 2 - See error 1. The control lines being wrong is less likely
:10517 : this time.
:10518 :
:10519 : Error 3 - The most likely problem now is the 2901 itself. The control
:10520 : lines and inputs are identical to error 1.
:10521 :
:10522 : Error 4 - See error 3.
:10523 :
:10524 :
:10525 :
U 1E80, B65E,15 :10526 T.E:-- MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
U 1E81, 3E80,15 :10527 MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
:10528 : ;BIT15 INDICATES START OF TEST TO CONSOLE
U 1E82, 10E0,15 :10529 MISC [SET.CP.ATTN] ;ISSUE CPU ATTN TO CONSOLE
:10530 :
WAIT.TE.0: :10531 JMP [WAIT.TE.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 1E83, 09E8,34 :10532 CLR LS[ERROR.MASK] ;CLEAR ERROR MASK.
U 1E84, E58A,15 :10533 CLR LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
U 1E85, 65A0,15 :10534 CLR WR[0]
U 1E86, 2F80,15 :10535 INC WR[0] ;SET WRO TO 1
U 1E87, 2040,15 :10536 MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
U 1E88, BE82,15 :10537 INC WR[0] ;SET WRO TO 2
U 1E89, 2040,15 :10538 MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
U 1E8A, 3E8C,15 :10539
LOOP.TE.1: :10540 CLR WR[1] ;SET WR1 TO 0
U 1E8B, 2F82,95 :10541 XOR WR[1] WITH LS[ZERO] TO Q ;PRELOAD Q REGISTER WITH ALL 0'S
U 1E8C, CD9C,95 :10542 MOV LS[ONES] TO WR[1] ;SET ALL BITS IN WR1
U 1E8D, 369E,95 :10543 XOR WR[1] WITH LS[ZERO] TO Q ;LOAD Q REGISTER WITH ALL 1'S
U 1E8E, CD9C,95 :10544 MOV Q TO WR[0] ;PUT RECEIVED DATA IN WRO
U 1E8F, A180,15 :10545 JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1E90, 0869,3C :10546 JMP [LOOP.TE.1] ;LOOP ON ERROR IF ENABLED
U 1E91, 89E8,B4 :10547 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
U 1E92, 8870,5C :10548
LOOP.TE.2: :10549 CLR WR[1] ;CLEAR WR1
U 1E93, 2F82,95 :10550 XOR WR[1] WITH LS[ZERO] TO Q ;LOAD Q REGISTER WITH ALL 0'S
U 1E94, CD9C,95 :10551 MOV Q TO WR[0] ;PUT RECEIVED DATA IN WRO
U 1E95, A180,15 :10552 JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1E96, 0869,3C :10553 JMP [LOOP.TE.2] ;LOOP ON ERROR IF ENABLED
U 1E97, 89E9,34 :10554 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 3
U 1E98, 8870,5C :10555 CLR LS[OS] ;CLEAR THE OS REGISTER
U 1E99, E5F8,15 :10556
LOOP.TE.3: :10557 CLR WR[1] ;CLEAR WR1
U 1E9A, 2F82,95 :10558 XOR WR[1] WITH LS[SHIFT.OS(4-0)] TO Q ;LOAD Q REGISTER WITH SHIFTING ONE PATTERN
U 1E9B, CDF6,95 :10558
```



```

U 1E9C, A180,15 :10559      MOV Q TO WR[0]          :PUT RECEIVED DATA IN WR0
U 1E9D, B6F6,95 :10560      MOV LS[SHIFT.OS(4-0)] TO WR[1] :PUT EXPECTED DATA IN WR1
U 1E9E, 0869,3C :10561      JSR [CHECK.RESULT]       :CHECK THE RESULT
U 1E9F, 89E9,A4 :10562      JMP [LOOP.TE.3]           :LOOP ON ERROR IF ENABLED
U 1EA0, 36F9,15 :10563      MOV LS[OS] TO WR[2]         :GET INDEX
U 1EA1, 2045,15 :10564      INC WR[2]                 :INCREMENT IT
U 1EA2, C52D,15 :10565      BIC LS[FFFFFF00] TO WR[2]    :CLEAR HIGH BITS
U 1EA3, BEF9,15 :10566      MOV WR[2] TO LS[OS]
U 1EA4, CD4B,35 :10567      XOR WR[2] WITH LS[#20] TO Q,    :IS IT EQUAL TO 32?
                        DT(LONG)&SET.ALU.CC
U 1EA5, 5B00,09 :10568      SKIP.IF[EQ]              :SKIP NEXT INSTRUCTION IF SO
U 1EA6, 89E9,A4 :10569      JMP [LOOP.TE.3]           :GO DO NEXT BIT
U 1EA7, 8870,5C :10570      JSR [INC.EN]                :INCREMENT ERROR NUMBER TO 4
                        :10571
                        LOOP.TE.4:
U 1EA8, 2F82,95 :10572      CLR WR[1]                  :CLEAR WR1
U 1EA9, DFF7,15 :10573      MCOM LS[SHIFT.OS(4-0)] TO WR[2] :SET UP SHIFTING 0 PATTERN
U 1EAA, BE11,15 :10574      MOV WR[2] TO LS[T8]          :PUT PATTERN IN LS
U 1EAB, 4D10,95 :10575      XOR WR[1] WITH LS[T8] TO Q    :LOAD Q REGISTER WITH SHIFTING ONE PATTERN
U 1EAC, A180,15 :10576      MOV Q TO WR[0]              :PUT RECEIVED DATA IN WR0
U 1EAD, A004,95 :10577      MOV WR[2] TO WR[1]          :PUT EXPECTED DATA IN WR1
U 1EAE, 0869,3C :10578      JSR [CHECK.RESULT]       :CHECK THE RESULT
U 1EAF, 09EA,84 :10579      JMP [LOOP.TE.4]           :LOOP ON ERROR IF ENABLED
U 1EB0, 36F9,15 :10580      MOV LS[OS] TO WR[2]         :GET INDEX
U 1EB1, 2045,15 :10581      INC WR[2]                 :INCREMENT IT
U 1EB2, C52D,15 :10582      BIC LS[FFFFFF00] TO WR[2]    :CLEAR HIGH BITS
U 1EB3, BEF9,15 :10583      MOV WR[2] TO LS[OS]
U 1EB4, CD4B,35 :10584      XOR WR[2] WITH LS[#20] TO Q,    :IS IT EQUAL TO 32?
                        DT(LONG)&SET.ALU.CC
U 1EB5, 5B00,09 :10585      SKIP.IF[EQ]              :SKIP NEXT INSTRUCTION IF SO
U 1EB6, 09EA,84 :10586      JMP [LOOP.TE.4]           :GO DO NEXT BIT
                        :10587
                        END.TE:
  
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST F - 2901 SRC=D.3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST F - 2901 SRC=D.Q Test (M8390 CPU Module)

Fiche 2 Frame D3

Sequence 235

Page 229

:10590 Page "TEST F - 2901 SRC=D.Q Test (M8390 CPU Module)"

:10591 ++

:10592

:10593

:10594

:10595

:10596

:10597

:10598

:10599

:10600

:10601

:10602

:10603

:10604

:10605

:10606

:10607

:10608

:10609

:10610

:10611

:10612

:10613

:10614

:10615

:10616

:10617

:10618

:10619

:10620

:10621

:10622

:10623

:10624

:10625

:10626

:10627

:10628

:10629

:10630

:10631

:10632

:10633

:10634

:10635

:10636

:10637

:10638

:10639

:10640

:10641

:10642

:10643

:10644

Test Description:

This test checks the source mode D.Q in the 2901. The macro instruction AND LS TO Q is used to write the Q reg in this mode. The control lines are as follows: SRC=D.Q FUNC=R AND S DST=LOAD.Q CIN=CIN. Only the SRC mode has not been previously tested. The data path is as follows: The Q reg is first written from LS. Then a AND with the Q reg and LS is performed. The result is read from the Q reg into WR 0, the expected result is loaded into WR 1, and the result is checked. Data patterns used are all zeros, all 1's, and alternating 1's and 0's. It is not necessary to use more patterns as only the SRC mode is being checked.

Assumptions:

It is assumed that the previous 2901 tests have run successfully, including the Q reg test.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load the Q reg with all 1's, load WR 0 with all 1's and execute AND LS TO Q with LS containing zeros.
- 3) Move the Q reg to WR 0, load WR 1 with expected result (all 0's), and check. This verifies that A.Q is not the source.
- 4) Load the Q reg with alternating 1's and 0's and execute AND LS TO Q with LS containing 1's.
- 5) Move Q to WR 0, load WR 1 with expected result (alternating 1's and 0's) and check. This verifies that O.Q is not the source.

Errors:

- Error 1 - SRC mode D.Q failed when ANDing 1's and 0's.
Error 2 - SRC mode D.Q failed when ANDING alternating 1's and 0's with all 1's.

Debug:

Error 1 - This error indicates a faulty 2901 or faulty control lines. Check the control lines first using the control line description at the beginning of this listing and the description of this test. If an error is found here during the AND instruction, check the DEST.CTL ROM and ALU.SRC.CTL PAL. The inputs for these IC's from CSR 22 through CSR 14 should be 1 0101 1001(B) respectively. If the control lines are OK, suspect a bad 2901.

Error 2 - Same as error 1, except a bad 2901 is more likely this time.

T.F: --

U 1EB7, B65E.15

MOV LS[BEGIN.TEST] TO WR[0]

;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD


```

U 1EB8, 3E80,15 :10645      MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
:10646                                     ; BIT15 INDICATES START OF TEST TO CONSOLE
U 1EB9, 10E0,15 :10647      MISC [SET.CP.ATTN] ;ISSUE CPU ATTN TO CONSOLE
:10648
U 1EBA, 09EB,A4 :10649      WAIT.TF.0: JMP [WAIT.TF.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 1EBB, E58A,15 :10650                                     ;CLEAR ERROR MASK.
U 1EBC, 65A0,15 :10651      CLR LS[ERROR.MASK] ;CLEAR PREVIOUS ERROR NUMBER
U 1EBD, 2F80,15 :10652      CLR LS[PREVIOUS.ERROR]
U 1EBE, 2040,15 :10653      CLR WR[0] ;SET WRO TO 1
U 1EBF, BE82,15 :10654      INC WR[0] ;SET ERROR NUMBER TO FIRST ERROR
U 1EC0, 2040,15 :10655      MOV WR[0] TO LS[ERROR.NUMBER] ;SET WRO TO 2
U 1EC1, 3E8C,15 :10656      INC WR[0] ;SET UP MODULE CODE FOR CPU MODULE
:10657      MOV WR[0] TO LS[MODULE.NUM]
U 1EC2, 589E,15 :10658      LOOP.TF.1: MOV LS[ONES] TO Q ;SET ALL BITS IN Q REGISTER
U 1EC3, B69E,15 :10659      MOV LS[ONES] TO WR[0] ;SET ALL BITS IN WRO
U 1EC4, 569C,15 :10660      AND LS[ZERO] TO Q ;AND ALL 0'S TO THE Q REGISTER
U 1EC5, A180,15 :10661      MOV Q TO WR[0] ;PUT RECEIVED DATA IN WRO
U 1EC6, 2F82,95 :10662      CLR WR[1] ;PUT EXPECTED DATA (0) IN WR1
U 1EC7, 0869,3C :10663      JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1EC8, 09EC,24 :10664      JMP [LOOP.TF.1] ;LOOP ON ERROR IF ENABLED
U 1EC9, 8870,5C :10665      JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
:10666
U 1ECA, D89A,15 :10667      LOOP.TF.2: MOV LS[ALTER.EVEN] TO Q ;SET PATTERN OF ALTERNATING BITS IN Q
U 1ECB, D69E,15 :10668      AND LS[ONES] TO Q ;AND ALL 1'S TO THE Q REGISTER
U 1ECC, A180,15 :10669      MOV Q TO WR[0] ;PUT RECEIVED DATA IN WRO
U 1ECD, B69A,95 :10670      MOV LS[ALTER.EVEN] TO WR[1] ;PUT EXPECTED DATA (#55555555) IN WR1
U 1ECE, 0869,3C :10671      JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1ECF, 89EC,A4 :10672      JMP [LOOP.TF.2] ;LOOP ON ERROR IF ENABLED
:10673      END.TF:

```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 10 - 2901 SRC= A.Q Test (M8390 CPU Module)
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAF module Rev 01.00
Fiche 2 Frame F3
Sequence 237 Page 231

Page "TEST 10 - 2901 SRC=A.Q Test (M8390 CPU Module)"

Test Description:

This test checks the source mode A.Q in the 2901. The macro instruction AND Q TO WR is used to use the Q reg in this mode. The control lines are as follows: SRC=A.Q FUNC=R AND S DST=WRITE.B.F CIN=NOCIN. Only the SRC mode has not been previously tested. The data path is as follows: The Q reg is first written from LS. Then a AND with the Q reg and WR 1 is performed. The result is written into WR 1 by the AND also. The received data is loaded into WR 0, the expected result is loaded into WR 1, and the result is checked. Data patterns used are all 1's and all 0's, and alternating 1's and 0's. It is not necessary to use more patterns as only the SRC mode is being checked.

Assumptions:

It is assumed that the previous 2901 tests have run successfully, including the Q reg test.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load the Q reg with all 1's, load WR 0 with all 1's and execute AND Q TO WR[1] with WR 1 containing zeros.
- 3) Check result for 0's. This verifies that D.Q is not the source.
- 4) Load the Q reg with alternating 1's and 0's and execute AND Q TO WR[1] with WR 1 containing 1's.
- 5) Load WR 1 with expected result (alternating 1's and 0's) and check. This verifies that 0.Q is not the source.

Errors:

- Error 1 - SRC mode A.Q failed when ANDING 1's and 0's.
Error 2 - SRC mode A.Q failed when ANDING alternating 1's and 0's with all 1's.

Debug:

Error 1 - This error indicates a faulty 2901 or faulty control lines. Check the control lines first using the control line description at the beginning of this listing and the description of this test. If an error is found here during the AND instruction, check the DEST.CTL ROM and ALU.SRC.CTL PAL. The inputs for these IC's from CSR 22 through CSR 14 should be 0 1011 0100(B) respectively. If the control lines are OK, suspect a bad 2901.

Error 2 - Same as error 1, except a bad 2901 is more likely this time.

U 1ED0, B65E,15

T.10:

MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD


```

U 1ED1, 3E80,15 :10729      MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
:10730      : BIT15 INDICATES START OF TEST TO CONSOLE
U 1ED2, 10E0,15 :10731      MISC [SET.CP.ATTN] ;ISSUE CPU ATTN TO CONSOLE
:10732      WAIT.T10.0:
U 1ED3, 09ED,34 :10733      JMP [WAIT.T10.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 1ED4, E58A,15 :10734      CLR LS[ERROR.MASK] ;CLEAR ERROR MASK.
U 1ED5, 65A0,15 :10735      CLR LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
U 1ED6, 2F80,15 :10736      CLR WR[0]
U 1ED7, 2040,15 :10737      INC WR[0] ;SET WRO TO 1
U 1ED8, BE82,15 :10738      MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
U 1ED9, 2040,15 :10739      INC WR[0] ;SET WRO TO 2
U 1EDA, 3E8C,15 :10740      MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
:10741      LOOP.T10.1:
U 1EDB, 589E,15 :10742      MOV LS[ONES] TO Q ;LOAD Q REGISTER WITH ALL 1'S
U 1EDC, B69E,15 :10743      MOV LS[ONES] TO WR[0] ;LOAD WRO WITH ALL 1'S
U 1EDD, 2F82,95 :10744      CLR WR[1] ;SET WR1 TO 0
U 1EDE, 2D02,95 :10745      AND Q TO WR[1] ;EXECUTE AND INSTRUCTION
U 1EDF, 2002,15 :10746      MOV WR[1] TO WR[0] ;PUT RECEIVED DATA IN WRO
U 1EE0, 2F82,95 :10747      CLR WR[1] ;PUT EXPECTED DATA (0) IN WR1
U 1EE1, 0869,3C :10748      JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1EE2, 89ED,B4 :10749      JMP [LOOP.T10.1] ;LOOP ON ERROR IF ENABLED
U 1EE3, 8870,5C :10750      JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
:10751      LOOP.T10.2:
U 1EE4, D89A,15 :10752      MOV LS[ALTER.EVEN] TO Q ;PUT ALTERNATING 1'S PATTERN IN Q REGISTER
U 1EE5, 369E,95 :10753      MOV LS[ONES] TO WR[1] ;SET ALL BITS IN WR1
U 1EE6, 2D02,95 :10754      AND Q TO WR[1] ;EXECUTE AND INSTRUCTION
U 1EE7, 2002,15 :10755      MOV WR[1] TO WR[0] ;PUT RECEIVED DATA IN WRO
U 1EE8, B69A,95 :10756      MOV LS[ALTER.EVEN] TO WR[1] ;PUT EXPECTED DATA (#55555555) IN WR1
U 1EE9, 0869,3C :10757      JSR [CHECK.RESULT] ;CHECK THE RESULT
U 1EEA, 89EE,44 :10758      JMP [LOOP.T10.2] ;LOOP ON ERROR IF ENABLED
:10759      END.T10:
  
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 11 - 2901 SRC=0.B Test (M8390 CPU Module)
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 11 - 2901 SRC=0.B Test (M8390 CPU Module)

Fiche 2 Frame H3

Sequence 239 Page 233
Rev 01.00

Page "TEST 11 - 2901 SRC=0.B Test (M8390 CPU Module)"

Test Description:

This test checks the source mode 0.B in the 2901. The macro instruction MCOM WR TO LS is used for this test. The control lines are as follows: SRC=0.B FUNC=R XNOR S DST=NOP CIN=NOCIN. Only the SRC mode has not been previously tested. The data path is as follows: WR 0 is written from LS, then MCOM WR TO LS is executed. This loads LS with the complement of WR 0. Then WR 0 is loaded with the result from LS, WR 1 is loaded with the expected result, and the result is checked. Data pattern used is alternating 1's and 0's. It is not necessary to use more patterns as only the SRC mode is being checked.

Assumptions:

It is assumed that the previous 2901 tests have run successfully.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load WR 0 with alternating 1's and 0's, and execute MCOM WR TO LS using a temporary location in LS to store the result.
- 3) Load WR 0 with result from LS, load WR 1 with expected result, and check.

Errors:

Error 1 - SRC mode 0.B failed.

Debug:

Error 1 - This error indicates a faulty 2901 or faulty control lines. Check the control lines first using the control line description at the beginning of this listing and the description of this test. If an error is found here during the MCOM instruction, check the DEST.CTL ROM and ALU.SRC.CTL PAL. The inputs for these IC's from CSR 22 through CSR 14 should be 1 1110 0000(B) respectively. If the control lines are OK, suspect a bad 2901.

T.11:

U 1EEB, B65E,15
U 1EEC, 3E80,15
U 1EED, 10E0,15
U 1EEE, 89EE,E4
U 1EEF, E58A,15
U 1EF0, 65A0,15
U 1EF1, 2F80,15
U 1EF2, 2040,15
U 1EF3, BE82,15

WAIT.T11.0:

```
MOV LS[BEGIN.TEST] TO WR[0]      ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS]  ;SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN]                ;BIT15 INDICATES START OF TEST TO CONSOLE
ISSUE CPU ATTN TO CONSOLE
JMP [WAIT.T11.0]                  ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR LS[ERROR.MASK]                ;CLEAR ERROR MASK.
CLR LS[PREVIOUS.ERROR]            ;CLEAR PREVIOUS ERROR NUMBER
CLR WR[0]
INC WR[0]                          ;SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER]     ;SET ERROR NUMBER TO FIRST ERROR
```


Page "TEST 12 - 2901 DST=WRITE.B.A Test (M8390 CPU Module)"

Test Description:

This test checks the destination mode WRITE.B.A in the 2901. The macro instruction SWAP LS WITH WR is used for this test. The control lines are as follows: SRC=D.0 FUNC=R OR S DST=WRITE.B.A CIN=NOCIN. Only the DST mode has not been previously tested. The data path is as follows: WR 0 is written with 0's from LS, then SWAP LS WITH WR is executed with LS containing 1's. The 0's from WR 0 are output on the Y BUS and written into LS, while the 1's in LS are written in WR 0. WR 0 is checked for 1's and LS is checked for 0's. Data patterns used are all 1's and all 0's. It is not necessary to use more patterns as only the DST mode is being checked.

Assumptions:

It is assumed that the previous 2901 tests have run successfully.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load WR 0 with 0's, and execute SWAP LS WITH WR using a temporary location in LS containing 1's.
- 3) Load WR 1 with expected result (all 1's) and check.
- 4) Move the LS location used in step 2 to WR 0, load WR 1 with expected result (all 0's) and check.

Errors:

- Error 1 - DST mode WRITE.B.A failed to write WR 0 from LS.
Error 2 - DST mode WRITE.B.A failed to output A and write into LS.

Debug:

Error 1 - This error indicates a faulty 2901 or faulty control lines. Check the control lines first using the control line description at the beginning of this listing and the description of this test. If an error is found here during the SWAP instruction, check the DEST.CTL ROM and ALU.SRC.CTL PAL. The inputs for these IC's from CSR 22 through CSR 14 should be 1 1001 0000(B) respectively. If the control lines are OK, suspect a bad 2901.

Error 2 - Same as error 1.

T.12:

U 1EFC, B65E,15
U 1EFD, 3E80,15
U 1EFE, 10E0,15
U 1EFF, 89EF,F4

```
MOV LS[BEGIN.TEST] TO WR[0]      ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS]  ;SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN]                ; BIT15 INDICATES START OF TEST TO CONSOLE
WAIT.T12.0:                        ;ISSUE CPU ATTN TO CONSOLE
JMP [WAIT.T12.0]                  ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
```


U 1F00, E58A,15	:10880	CLR LS[ERROR.MASK]	:CLEAR ERROR MASK.
U 1F01, 65A0,15	:10881	CLR LS[PREVIOUS.ERROR]	:CLEAR PREVIOUS ERROR NUMBER
U 1F02, 2F80,15	:10882	CLR WR[0]	
U 1F03, 2040,15	:10883	INC WR[0]	:SET WRO TO 1
U 1F04, BE82,15	:10884	MOV WR[0] TO LS[ERROR.NUMBER]	:SET ERROR NUMBER TO FIRST ERROR
U 1F05, 2040,15	:10885	INC WR[0]	:SET WRO TO 2
U 1F06, 3E8C,15	:10886	MOV WR[0] TO LS[MODULE.NUM]	:SET UP MODULE CODE FOR CPU MODULE
	:10887	LOOP.T12.1:	
U 1F07, 2F80,15	:10888	CLR WR[0]	:LOAD WRO WITH 0
U 1F08, 369E,95	:10889	MOV LS[ONES] TO WR[1]	
U 1F09, BE10,95	:10890	MOV WR[1] TO LS[T8]	:SET LS TO 1'S
U 1FOA, 6410,15	:10891	SWAP LS[T8] WITH WR[0]	:EXECUTE SWAP
U 1FOB, 0869,3C	:10892	JSR [CHECK.RESULT]	:CHECK THE RESULT
U 1FOC, 89F0,74	:10893	JMP [LOOP.T12.1]	:LOOP ON ERROR IF ENABLED
U 1FOD, B643,15	:10894	MOV LS[#2] TO WR[2]	
U 1FOE, 3E83,15	:10895	MOV WR[2] TO LS[ERROR.NUMBER]	:SET ERROR NUMBER TO 2
U 1FOF, B610,15	:10896	MOV LS[T8] TO WR[0]	:GET RECEIVED DATA
U 1F10, 2F82,95	:10897	CLR WR[1]	:SET EXPECTED DATA
U 1F11, 0869,3C	:10898	JSR [CHECK.RESULT]	:CHECK THE RESULT
U 1F12, 89F0,74	:10899	JMP [LOOP.T12.1]	:LOOP ON ERROR IF ENABLED
	:10900	END.T12:	

Page "TEST 13 - 2901 RAM Shift Test (M8390 CPU Module)"

++

Test Description:

This test checks the 2901 RAM shift destination mode, at speed, for both left and right shifts. RAM shifts have been checked previously in single step mode by the console, but this is the first shift test run at speed. The macro instructions ROL WR and ROR WR will be used for this test. This will check the shift capability of the 2901, the shift outputs, and part of the SHIFT DATA PAL. The 2901 control lines for the ROL WR is as follows: SRC=0.B FUNC=R OR S DST=LSHF.RAM CIN=CIN. The value of SI (shift input portion of EXTENDED micro-word) used as an input to the SHIFT DATA PAL is 0. The 2901 control lines for the ROR macro-instruction are the same as ROL except FUNC=RSHF.RAM. The data path is as follows: The Q reg and WR 0 are written with a data pattern from LS, and the ROL or ROR instruction is issued. WR 1 is written with the expected data and the result is checked. Then the Q reg is moved to WR 0 and WR 1 is written with the original data and the result checked. This assures that the Q reg is not getting shifted. The data patterns used are shifting 1's and shifting 0's.

Assumptions:

It is assumed that the previous 2901 tests Have run successfully. This is the first 2901 shift test run at speed.

Test Steps:

- 1) Set up error mask\$Error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load Q reg and WR 0 with a shifting 1 pattern.
- 3) Execute ROL WR[0], load WR[1] with expected result, and check.
- 4) Move the contents of the Q reg to WR[0], load WR[1] with original data and check that Q reg was not altered.
- 5) Repeat steps 2 through 4 until a one has been shifted into bit 31 and checked.
- 6) Repeat steps 2 through 4 with a shifting 0 pattern until a 0 has been shifted into bit 31 and checked.
- 7) Load Q reg and WR 0 with a shifting 1 pattern (starting at bit 31).
- 8) Execute ROR WR[0], load WR[1] with expected result, and check.
- 9) Move the contents of the Q reg to WR[0], load WR[1] with original data and check that Q reg was not altered.
- 10) Repeat steps 7 through 9 until a one has been shifted into bit 0 and checked.
- 11) Repeat steps 7 through 9 with a shifting 0 pattern until a 0 has been shifted into bit 0 and checked.

Errors:

- Error 1 - ROL WR[0] failed to shift a single 1 bit correctly.
- Error 2 - ROL WR[0] altered Q register.
- Error 3 - ROL WR[0] failed to shift a single 0 bit correctly.
- Error 4 - ROL WR[0] altered Q register.
- Error 5 - ROR WR[0] failed to shift a single 1 bit correctly.

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 13 - 2901 RAM M 3
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module Sequence 244 Page 238
TEST 13 - 2901 RAM Shift Test (M8390 CPU Module)

:10956 : Error 6 - ROR WR[0] altered Q register.
:10957 : Error 7 - ROR WR[0] failed to shift a single 0 bit correctly.
:10958 : Error 8 - ROR WR[0] altered Q register.
:10959 :
:10960 : Debug:
:10961 :
:10962 : Error 1 - If only bit 0 is failing, suspect the SHIFT DATA PAL or
:10963 : the signal RAM SHF MSB H out of the last (upper) 2901. During the
:10964 : ROL instruction, the inputs to the SHIFT DATA PAL should be high
:10965 : on DEST CTL 1 H and DEST CTL 2 H, low on CSR 13 through CSR 11, and
:10966 : the signal RAM SHF MSB H should be low. The output RAM SHF LSB H on
:10967 : pin 16 should be low. If bits other than bit 0 are failing, suspect
:10968 : a bad 2901 or bad control lines. Refer to the 2901 control line
:10969 : description at the beginning of this listing and this test description
:10970 : and check the control lines at the 2901. If these are incorrect, then
:10971 : check the DEST CTL ROM and the SRC.ALU CTL PAL. The inputs to these
:10972 : IC's for CSR 22 through 14 should be 0 1000 1111(B) respectively. If
:10973 : the control lines are OK, suspect a bad 2901.
:10974 :
:10975 : Error 2 - This error indicates that the Q reg is being shifted on a
:10976 : LSHF.RAM destination. Check the 2901 control lines first, paying
:10977 : particular attention to the DEST control signals. See error 1 for
:10978 : control line information. If the control lines are OK, then suspect
:10979 : the 2901 itself.
:10980 :
:10981 : Error 3 - See error 1. The only difference is that RAM SHF MSB H
:10982 : should be high and the output RAM SHF LSB H on pin 16 should be high.
:10983 :
:10984 : Error 4 - Same as error 2.
:10985 :
:10986 : Error 5 - If only bit 31 is failing, suspect the SHIFT DATA PAL or
:10987 : the signal RAM SHF LSB H out of the low order 2901. During the
:10988 : ROR instruction, the inputs to the SHIFT DATA PAL should be low on DEST
:10989 : CTL 1 H and high on DEST CTL 2 H, low on CSR 13 through CSR 11, and
:10990 : the signal RAM SHF LSB H should be low. The output RAM SHF MSB H on
:10991 : pin 18 should be low. If bits other than bit 0 are failing, suspect
:10992 : a bad 2901 or bad control lines. Refer to the 2901 control line
:10993 : description at the beginning of this listing and this test description
:10994 : and check the control lines at the 2901. If these are incorrect, then
:10995 : check the DEST CTL ROM and the SRC.ALU CTL PAL. The inputs to these
:10996 : IC's for CSR 22 through 14 should be 0 1000 1101(B) respectively. If
:10997 : the control lines are OK, suspect a bad 2901.
:10998 :
:10999 : Error 6 - This error indicates that the Q reg is being shifted on a
:11000 : RSHF.RAM destination. Check the 2901 control lines first, paying
:11001 : particular attention to the DEST control signals. See error 5 for
:11002 : control line information. If the control lines are OK, then suspect
:11003 : the 2901 itself.
:11004 :
:11005 : Error 7 - See error 5. The only difference is that RAM SHF LSB H
:11006 : should be high and the output RAM SHF MSB H on pin 18 should be high.
:11007 :
:11008 : Error 8 - Same as error 6.
:11009 :
:11010 : --

```

T.13:
U 1F13, B65E,15 :11011
U 1F14, 3E80,15 :11012
U 1F15, 10E0,15 :11013
U 1F16, 89F1,64 :11014
U 1F17, E58A,15 :11015
U 1F18, 65A0,15 :11016
U 1F19, 2F80,15 :11017
U 1F1A, 2040,15 :11018
U 1F1B, BE82,15 :11019
U 1F1C, 2040,15 :11020
U 1F1D, 3E8C,15 :11021
U 1F1E, E5F8,15 :11022
U 1F1F, B640,15 :11023
U 1F20, 5D4A,15 :11024
U 1F21, 3E10,15 :11025
U 1F22, 36F6,15 :11026
U 1F23, AAC0,15 :11027
U 1F24, A001,95 :11028
U 1F25, A3C0,15 :11029
U 1F26, 36F9,15 :11030
U 1F27, 2045,15 :11031
U 1F28, C52D,15 :11032
U 1F29, BEF9,15 :11033
U 1F2A, B6F6,95 :11034
U 1F2B, F612,15 :11035
U 1F2C, 0869,3C :11036
U 1F2D, 09F2,24 :11037
U 1F2E, D812,15 :11038
U 1F2F, B642,95 :11039
U 1F30, 3E82,95 :11040
U 1F31, A180,15 :11041
U 1F32, 2006,95 :11042
U 1F33, F612,15 :11043
U 1F34, 0869,3C :11044
U 1F35, 09F2,24 :11045
U 1F36, D812,15 :11046
U 1F37, CD11,35 :11047
U 1F38, 09F2,21 :11048
U 1F39, 8870,5C :11049
U 1F3A, E5F8,15 :11050
U 1F3B, 5FF6,15 :11051
U 1F3C, AAC0,15 :11052
U 1F3D, A001,95 :11053
U 1F3E, A3C0,15 :11054
U 1F3F, 36F9,15 :11055
U 1F40, 2045,15 :11056
U 1F41, C52D,15 :11057
U 1F42, BEF9,15 :11058

T.13:
    MOV LS[BEGIN.TEST] TO WR[0]
    MOV WR[0] TO LS[CONTROL.STATUS]
    MISC [SET.CP.ATTN]
    WAIT.T13.0:
        JMP [WAIT.T13.0]
        CLR LS[ERROR.MASK]
        CLR LS[PREVIOUS.ERROR]
        CLR WR[0]
        INC WR[0]
        MOV WR[0] TO LS[ERROR.NUMBER]
        INC WR[0]
        MOV WR[0] TO LS[MODULE.NUM]
        CLR LS[OS]
        MOV LS[#1] TO WR[0]
        SUB WR[0] FROM LS[#20] TO WR
        MOV WR[0] TO LS[T8]
    LOOP.T13.1:
        MOV LS[SHIFT.OS(4-0)] TO WR[0]
        MOV WR[0] TO Q
        MOV WR[0] TO WR[3]
        ROL WR[0]
        MOV LS[OS] TO WR[2]
        INC WR[2]
        BIC LS[FFFFFFF00] TO WR[2]
        MOV WR[2] TO LS[OS]
        MOV LS[SHIFT.OS(4-0)] TO WR[1]
        MOV Q TO LS[T9]
        JSR [CHECK.RESULT]
        JMP [LOOP.T13.1]
        MOV LS[T9] TO Q
        MOV LS[#2] TO WR[1]
        MOV WR[1] TO LS[ERROR.NUMBER]
    T13.2:
        MOV Q TO WR[0]
        MOV WR[3] TO WR[1]
        MOV Q TO LS[T9]
        JSR [CHECK.RESULT]
        JMP [LOOP.T13.1]
        MOV LS[T9] TO Q
        XOR WR[2] WITH LS[T8] TO Q,
        DT(LONG)&SET.ALU.CC
        JMP.IF[NEQ] TO [LOOP.T13.1]
        JSR [INC.EN]
        CLR LS[OS]
    LOOP.T13.3:
        MCOM LS[SHIFT.OS(4-0)] TO WR[0]
        MOV WR[0] TO Q
        MOV WR[0] TO WR[3]
        ROL WR[0]
        MOV LS[OS] TO WR[2]
        INC WR[2]
        BIC LS[FFFFFFF00] TO WR[2]
        MOV WR[2] TO LS[OS]

```

```

:SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
:SET BIT15 IN CNTROL AND STATUS WORD
:BIT15 INDICATES START OF TEST TO CONSOLE
:ISSUE CPU ATTN TO CONSOLE
:LOOP HERE WAITING FOR CONSOLE TO RESPOND
:CLEAR ERROR MASK.
:CLEAR PREVIOUS ERROR NUMBER
:SET WRO TO 1
:SET ERROR NUMBER TO FIRST ERROR
:SET WRO TO 2
:SET UP MODULE CODE FOR CPU MODULE
:CLEAR THE OS REGISTER
:SET WRO TO 1
:SET WRO TO 31
:T8=31
:SET UP SHIFTING ONE PATTERN IN WRO
:AND IN Q REGISTER
:ALSO SAVE IT IN WR3
:EXECUTE THE ROTATE LEFT
:GET THE INDEX
:INCREMENT IT
:CLEAR HIGH BITS
:PUT EXPECTED DATA IN WR1
:SAVE THE Q REGISTER
:CHECK THE RESULT
:LOOP ON ERROR IF ENABLED
:RESTORE Q
:SET ERROR NUMBER TO 2
:GET Q REGISTER
:GET EPECTED DATA
:SAVE THE Q REGISTER
:CHECK THAT Q WAS NOT SHIFTED
:LOOP ON ERROR IF ENABLED
:RESTORE Q
:IS IT EQUAL TO 31
:IF NOT GO DO NEXT BIT
:INCREMENT ERROR NUMBER TO 3
:RESET INDEX
:SET UP SHIFTING ZERO PATTERN IN WRO
:AND IN Q REGISTER
:ALSO SAVE IT IN WR3
:EXECUTE THE ROTATE LEFT
:GET THE INDEX
:INCREMENT IT
:CLEAR HIGH BITS

```


ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

TEST 13 - 2901 RAM 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module Rev 01.00 Page 240
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 13 - 2901 RAM Shift Test (M8390 CPU Module)

```
U 1F43, DFF6.95 :11066 MCOM LS[SHIFT.OS(4-0)] TO WR[1] :PUT EXPECTED DATA IN WR1
U 1F44, F612.15 :11067 MOV Q TO LS[T9] :SAVE THE Q REGISTER
U 1F45, 0869.3C :11068 JSR [CHECK.RESULT] :CHECK THE RESULT
U 1F46, 89F3.B4 :11069 JMP [LOOP.T13.3] :LOOP ON ERROR IF ENABLED
U 1F47, D812.15 :11070 MOV LS[T9] TO Q :RESTORE Q
U 1F48, B644.95 :11071 MOV LS[#4] TO WR[1]
U 1F49, 3E82.95 :11072 MOV WR[1] TO LS[ERROR.NUMBER] :SET ERROR NUMBER TO 4
:11073
T13.4:
U 1F4A, A180.15 :11074 MOV Q TO WR[0] :GET Q REGISTER
U 1F4B, 2006.95 :11075 MOV WR[3] TO WR[1] :GET EXPECTED DATA
U 1F4C, F612.15 :11076 MOV Q TO LS[T9] :SAVE THE Q REGISTER
U 1F4D, 0869.3C :11077 JSR [CHECK.RESULT] :CHECK THAT Q WAS NOT SHIFTED
U 1F4E, 89F3.B4 :11078 JMP [LOOP.T13.3] :LOOP ON ERROR IF ENABLED
U 1F4F, D812.15 :11079 MOV LS[T9] TO Q :RESTORE Q
:11080 XOR WR[2] WITH LS[18] TO Q, :IS IT EQUAL TO 31
:11081 DT(LONG)&SET.ALU.CC
U 1F50, CD11.35 :11082 JMP.[IF[NEQ] TO [LOOP.T13.3] :IF NOT GO DO NEXT BIT
U 1F51, 89F3.B1 :11083 JSR [INC.EN] :INCREMENT ERROR NUMBER TO 5
:11084
LOOP.T13.5:
U 1F53, 36F6.15 :11085 MOV LS[SHIFT.OS(4-0)] TO WR[0] :SET UP SHIFTING 1 PATTERN IN WRO
U 1F54, AAC0.15 :11086 MOV WR[0] TO Q :AND IN THE Q REGISTER
U 1F55, A001.95 :11087 MOV WR[0] TO WR[3] :ALSO IN WR3
U 1F56, 2340.15 :11088 ROR WR[0] :EXECUTE ROTATE RIGHT
U 1F57, 2F85.15 :11089 CLR WR[2] :WR2=0
U 1F58, 2045.15 :11090 INC WR[2] :WR2=1
U 1F59, DDF9.15 :11091 SUB WR[2] FROM LS[OS] TO WR :DECREMENT INDEX AND STORE IN WR2
U 1F5A, C52D.15 :11092 BIC LS[FFFFFFF0] TO WR[2]
U 1F5B, BEF9.15 :11093 MOV WR[2] TO LS[OS]
U 1F5C, B6F6.95 :11094 MOV LS[SHIFT.OS(4-0)] TO WR[1] :PUT EXPECTED DATA IN WR1
U 1F5D, F612.15 :11095 MOV Q TO LS[T9] :SAVE THE Q REGISTER
U 1F5E, 0869.3C :11096 JSR [CHECK.RESULT] :CHECK THE RESULT
U 1F5F, 09F5.34 :11097 JMP [LOOP.T13.5] :LOOP ON ERROR IF ENABLED
U 1F60, D812.15 :11098 MOV LS[T9] TO Q :RESTORE Q
U 1F61, B644.95 :11099 MOV LS[#4] TO WR[1]
U 1F62, C742.95 :11100 BIS LS[#2] TO WR[1] :WR1=6
U 1F63, 3E82.95 :11101 MOV WR[1] TO LS[ERROR.NUMBER] :SET ERROR NUMBER TO 6
:11102
T13.6:
U 1F64, A180.15 :11103 MOV Q TO WR[0] :GET Q
U 1F65, 2006.95 :11104 MOV WR[3] TO WR[1] :GET EXPECTED DATA
U 1F66, F612.15 :11105 MOV Q TO LS[T9] :SAVE THE Q REGISTER
U 1F67, 0869.3C :11106 JSR [CHECK.RESULT] :CHECK THAT Q WAS NOT SHIFTED
U 1F68, 09F5.34 :11107 JMP [LOOP.T13.5] :LOOP ON ERROR IF ENABLED
U 1F69, D812.15 :11108 MOV LS[T9] TO Q :RESTORE Q
:11109 XOR WR[2] WITH LS[ZERO] TO Q, :IS IT EQUAL TO 0?
:11110 DT(LONG)&SET.ALU.CC
U 1F6A, 4D9D.35 :11111 JMP.[IF[NEQ] TO [LOOP.T13.5] :IF NOT GO DO NEXT BIT
U 1F6B, 09F5.31 :11112 JSR [INC.EN] :INCREMENT ERROR NUMBER TO 7
U 1F6C, 8870.5C :11113 MOV LS[T8] TO WR[1]
U 1F6D, 3610.95 :11114 MOV WR[1] TO LS[OS] :SET OS TO 31
:11115
LOOP.T13.7:
U 1F6F, 5FF6.15 :11116 MCOM LS[SHIFT.OS(4-0)] TO WR[0] :SET UP SHIFTING 0 PATTERN IN WRO
U 1F70, AAC0.15 :11117 MOV WR[0] TO Q :AND IN THE Q REGISTER
U 1F71, A001.95 :11118 MOV WR[0] TO WR[3] :ALSO IN WR3
U 1F72, 2340.15 :11119 ROR WR[0] :EXECUTE ROTATE RIGHT
U 1F73, 2F85.15 :11120 CLR WR[2]
```

U 1F74, 2045,15	:11121	INC WR[2]	:WR2=1
U 1F75, DDF9,15	:11122	SUB WR[2] FROM LS[OS] TO WR	:DECREMENT INDEX AND STORE IN WR2
U 1F76, C52D,15	:11123	BIC LS[FFFFFF00] TO WR[2]	
U 1F77, BEF9,15	:11124	MOV WR[2] TO LS[OS]	
U 1F78, DFF6,95	:11125	MCOM LS[SHIFT.OS(4-0)] TO WR[1]	:PUT EXPECTED DATA IN WR1
U 1F79, F612,15	:11126	MOV Q TO LS[T9]	:SAVE THE Q REGISTER
U 1F7A, 0869,3C	:11127	JSR [CHECK.RESULT]	:CHECK THE RESULT
U 1F7B, 09F5,34	:11128	JMP [LOOP.T13.5]	:LOOP ON ERROR IF ENABLED
U 1F7C, D812,15	:11129	MOV LS[T9] TO Q	:RESTORE Q
U 1F7D, 3646,95	:11130	MOV LS[8] TO WR[1]	
U 1F7E, 3E82,95	:11131	MOV WR[1] TO LS[ERROR.NUMBER]	:SET ERROR NUMBER TO 8
	:11132		
U 1F7F, A180,15	:11133	MOV Q TO WR[0]	:GET Q
U 1F80, 2006,95	:11134	MOV WR[3] TO WR[1]	:GET EXPECTED DATA
U 1F81, F612,15	:11135	MOV Q TO LS[T9]	:SAVE THE Q REGISTER
U 1F82, 0869,3C	:11136	JSR [CHECK.RESULT]	:CHECK THAT Q WAS NOT SHIFTED
U 1F83, 09F6,F4	:11137	JMP [LOOP.T13.7]	:LOOP ON ERROR IF ENABLED
U 1F84, D812,15	:11138	MOV LS[T9] TO Q	:RESTORE Q
	:11139	XOR WR[2] WITH LS[ZERO] TO Q,	:IS IT EQUAL TO 0?
		DT(LONG)&SET.ALU.CC	
U 1F85, 4D9D,35	:11140	JMP.[IF[NEQ] TO [LOOP.T13.7]	:IF NOT GO DO NEXT BIT
U 1F86, 09F6,F1	:11141	JMP [END.T13]	
U 1F87, 0B02,04	:11142		
	:11143		
	:11144		

T13.8:
 3020:
 END.T13:

Page "TEST 14 - 2901 RAM and Q Shift Test (M8390 CPU Module)"

Test Description:

This test checks the 2901 RAM and Q shift destination mode for both left and right shifts on the RAM and Q registers combined. The macro instructions ROLC WR.Q and RORC WR.Q will be used for this test. This will check the shift capability of the 2901, the shift outputs, and part of the SHIFT DATA PAL. The 2901 control lines for the ROLC WR.Q is as follows: SRC=0.B FUNC=R OR S DST=LSHF.RAM.Q CIN=CIN. The value of SI (shift input portion of EXTENDED micro-word) used as an input to the SHIFT DATA PAL is 1. The 2901 control lines for the RORC macro-instruction are the same as ROLC except FUNC=RSHF.RAM.Q and CIN=NOCIN. The data path is as follows: The Q reg and WR 0 are written with a data pattern from LS, and the ROLC or RORC instruction is issued. WR 1 is written with the expected data and the result is checked. Then the Q reg is moved to WR 0 and WR 1 is written with the expected data and the result checked. The data patterns used are shifting 1's and shifting 0's.

Assumptions:

It is assumed that the previous 2901 shift test has run successfully. This is the first test of the Q reg shift.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load Q reg and WR 0 with a shifting 1 pattern. Start with a 1 in bit 0 of the Q reg and 0's in WR 0.
- 3) Execute ROLC WR[0].Q, load WR[1] with expected result, and check.
- 4) Move the contents of the Q reg to WR[0], load WR[1] with expected data and check.
- 5) Repeat steps 2 through 4 until a one has been shifted into bit 31 of WR 0 and checked (63 shifts).
- 6) Repeat steps 2 through 4 with a shifting 0 pattern starting with a 0 in bit 0 of the Q reg and 1's in the other bits and in WR 0. Repeat until a 0 has been shifted into bit 31 of WR 0 and checked.
- 7) Load Q reg and WR 0 with a shifting 1 pattern. Start with a 1 in bit 31 of WR 0 and 0's in the Q reg.
- 8) Execute RORC WR[0].Q, load WR[1] with expected result, and check.
- 9) Move the contents of the Q reg to WR[0], load WR[1] with expected data and check.
- 10) Repeat steps 7 through 9 until a one has been shifted into bit 0 of the Q reg and checked (63 shifts).
- 11) Repeat steps 7 through 9 with a shifting 0 pattern starting with a 0 in bit 31 of WR 0 and 1's in the other bits and in the Q reg. Repeat until a 0 has been shifted into bit 0 of the Q reg and checked.

Errors:

OTHER data : Shifting 1 pattern utilized

:11200
:11201
:11202
:11203
:11204
:11205
:11206
:11207
:11208
:11209
:11210
:11211
:11212
:11213
:11214
:11215
:11216
:11217
:11218
:11219
:11220
:11221
:11222
:11223
:11224
:11225
:11226
:11227
:11228
:11229
:11230
:11231
:11232
:11233
:11234
:11235
:11236
:11237
:11238
:11239
:11240
:11241
:11242
:11243
:11244
:11245
:11246
:11247
:11248
:11249
:11250
:11251
:11252
:11253
:11254

Error 1 - ROLC WR[0].Q failed to shift WR 0 correctly. Data is shifting 1's.
Error 2 - ROLC WR[0].Q failed to shift Q reg correctly. Data is shifting 1's.
Error 3 - ROLC WR[0].Q failed to shift WR 0 correctly. Data is shifting 0's.
Error 4 - ROLC WR[0].Q failed to shift Q reg correctly. Data is shifting 0's.
Error 5 - RORC WR[0].Q failed to shift WR 0 correctly. Data is shifting 1's.
Error 6 - RORC WR[0].Q failed to shift Q reg correctly. Data is shifting 1's.
Error 7 - RORC WR[0].Q failed to shift WR 0 correctly. Data is shifting 0's.
Error 8 - RORC WR[0].Q failed to shift Q reg correctly. Data is shifting 0's.

Debug:

Error 1 - If only bit 0 is failing, suspect the SHIFT DATA PAL or the signal Q SHF MSB H out of the (last (upper) 2901. During the ROLC instruction, the inputs to the SHIFT DATA PAL should be high on DEST CTL 1 H and DEST CTL 2 H, low on CSR 13 and 12, high on CSR 11, and the signal Q SHF MSB H should be low. The output RAM SHF LSB H on pin 16 should be low. If bits other than bit 0 are failing, suspect a bad 2901 or bad control lines. Refer to the 2901 control line description at the beginning of this listing and this test description and check the control lines at the 2901. If these are incorrect, then check the DEST CTL ROM and the SRC.ALU CTL PAL. The inputs to these IC's for CSR 22 through 14 should be 0 1000 1110(B) respectively. If the control lines are OK, suspect a bad 2901.

Error 2 - If only bit 0 is failing, suspect the SHIFT DATA PAL or the 2901. The Q reg shift input, Q SHF LSB H on pin 21 has not been used in previous tests. If bit 0 is supposed to be a 0, check for a low on this input. If bit 0 is supposed to be a 1, check for a high. During the ROLC instruction, the inputs to the SHIFT DATA PAL should be high on DEST CTL 1 H and DEST CTL 2 H, low on CSR 13 and 12, high on CSR 11, and the signal RAM SHF MSB H is dependant on bit 31 of WR 0. The output Q SHF LSB H on pin 15 should follow bit 31 of WR 0. If Q SHF LSB H at the 2901 is OK, suspect a bad 2901 or bad control lines. Refer to the 2901 control line description at the beginning of this listing and this test description and check the control lines at the 2901. If these are incorrect, then check the DEST CTL ROM and the SRC.ALU CTL PAL. The inputs to these IC's for CSR 22 through 14 should be 0 1000 1110(B) respectively. If the control lines are OK, suspect a bad 2901.

Error 3 - See error 1. The only difference is that Q SHF MSB H should be high and the output RAM SHF LSB H out of the SHIFT DATA PAL should be high.

Error 4 - Same as error 2.

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 14 - 2901 RAM and Q Shift Test (M8390 CPU Module)
MICRO2 1L(03) 3-MAR-82 09:34:33
F 4
3-MAR-82
Fiche 2 Frame F4
ENKCA Micro-diagnostic for DAP module
Sequence 250
Rev 01.00
Page 244

:11255 : Error 5 - If only bit 31 is failing, suspect the SHIFT DATA PAL or the
:11256 : 2901. The Q reg shift output, Q SHF LSB H on pin 21 has not been used
:11257 : in previous tests. If bit 31 is supposed to be a 0, check for a low
:11258 : on this output. If bit 31 is supposed to be a 1, check for a high.
:11259 : During the RORC instruction, the inputs to the SHIFT DATA PAL should
:11260 : be low on DEST CTL 1 H, high on DEST CTL 2 H, low on CSR 13 and 12,
:11261 : high on CSR 11, and the signal RAM SHF MSB H is dependant on bit 0 of
:11262 : the Q reg. The output RAM SHF MSB H on pin 18 should follow bit 0 of
:11263 : the Q reg. If RAM SHF MSB H at the 2901 is OK, suspect a bad 2901 or
:11264 : bad control lines. Refer to the 2901 control line description at the
:11265 : beginning of this listing and this test description and check the
:11266 : control lines at the 2901. If these are incorrect, then check the
:11267 : DEST CTL ROM and the SRC.ALU CTL PAL. The inputs to these IC's for
:11268 : CSR 22 through 14 should be 0 1000 1100(B) respectively. If the
:11269 : control lines are OK, suspect a bad 2901.

:11270 :
:11271 : Error 6 - If only bit 31 is failing, suspect the SHIFT DATA PAL or
:11272 : the signal Q SHF MSB H input to the high (upper) 2901. During the
:11273 : RORC instruction, the inputs to the SHIFT DATA PAL should be low on
:11274 : DEST CTL 1 H, high on DEST CTL 2 H, low on CSR 13 and 12, high on CSR
:11275 : 11, and the signal RAM SHF LSB H should be low. The output Q SHF MSB
:11276 : H on pin 17 should be low. If bits other than bit 31 are failing,
:11277 : suspect A bad 2901 or bad control lines. Refer to the 2901 control
:11278 : line description at the beginning of this listing and this test
:11279 : description and check the control lines at the 2901. If these are
:11280 : incorrect, then check the DEST CTL ROM and the SRC.ALU CTL PAL. The
:11281 : inputs to these IC's for CSR 22 through 14 should be 0 1000 1100(B)
:11282 : respectively. If the control lines are OK, suspect a bad 2901.

:11283 :
:11284 : Error 7 - See error 5. The only difference is that Q SHF LSB H should
:11285 : be high and the output RAM SHF MSB H out of the SHIFT DATA PAL should
:11286 : be high.

:11287 :
:11288 : Error 8 - Same as error 6, except RAM SHF LSB H should be high and the
:11289 : output Q SHF MSB H on pin 17 should be high.

:11290 :
:11291 : Note: In the error printouts WR 3 (the DATA pointer) will be printed
:11292 : under the heading OTHER.

:11293 :
:11294 :
:11295 :
:11296 :
:11297 : T.14:

U 3020, B65E,15 :11298
U 3021, 3E80,15 :11299
U 3022, 10E0,15 :11300
U 3023, 0B02,34 :11301
U 3024, E58A,15 :11302
U 3025, 65A0,15 :11303
U 3026, 2F80,15 :11304
U 3027, 2040,15 :11305
U 3028, BE82,15 :11306
U 3029, A3C0,15 :11307

WAIT.T14.0:

MOV LS[BEGIN.TEST] TO WR[0] :SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] :SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN] :BIT15 INDICATES START OF TEST TO CONSOLE
:ISSUE CPU ATTN TO CONSOLE
JMP [WAIT.T14.0] :LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR LS[ERROR.MASK] :CLEAR ERROR MASK.
CLR LS[PREVIOUS.ERROR] :CLEAR PREVIOUS ERROR NUMBER
CLR WR[0]
INC WR[0] :SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER] :SET ERROR NUMBER TO FIRST ERROR
ROL WR[0] :SET WRO TO 2

```

U 302A, 3E8C,15 :11310      MOV WR[0] TO LS[MODULE.NUM]      ;SET UP MODULE CODE FOR CPU MODULE
U 302B, B670,15 :11311      MOV LS[BIT24] TO WR[0]
U 302C, 3E80,15 :11312      MOV WR[0] TO LS[CONTROL.STATUS] ;"OTHER" DATA WILL CONTAIN CONTENTS OF WR3
U 302D, 364B,15 :11313      MOV LS[#20] TO WR[2]
U 302E, 2105,15 :11314      DEC WR[2]
U 302F, 3E13,15 :11315      MOV WR[2] TO LS[T9]
U 3030, E5F8,15 :11316      CLR LS[OS]
U 3031, 2F87,95 :11317      CLR WR[3]
                        :WR3 WILL BE USED AS A POINTER FOR THE DATA
                        : IF WR3=ZERO DATA IS IN Q
                        : IF WR3=ONES DATA IS IN WRO
U 3032, BE39,95 :11318
                        :11319
                        :11320      MOV WR[3] TO LS[ADDRESS.DATA]
LOOP.T14.1:
U 3033, 3641,15 :11321      MOV LS[#1] TO WR[2]
U 3034, 3E83,15 :11322      MOV WR[2] TO LS[ERROR.NUMBER] ;ERROR # = 1
                        :11323
                        :11324      TST WR[3]
U 3035, 2007,B5 :11325      DT(LONG)&SET.ALU.CC ;WR3 SET?
U 3036, 0B03,A9 :11326      JMP.IF[EQL] TO [T14.SET.Q.1] ;JUMP IF NOT
U 3037, 36F6,15 :11327      MOV LS[SHIFT.OS(4-0)] TO WR[0] ;PUT DATA IN WRO
U 3038, AB80,15 :11328      CLR Q
U 3039, 8B03,C4 :11329      JMP [T14.ROTATE.1] ;GO ROTATE
                        :11330
T14.SET.Q.1:
U 303A, D8F6,15 :11331      MOV LS[SHIFT.OS(4-0)] TO Q ;PUT DATA IN Q
U 303B, 2F80,15 :11332      CLR WR[0]
                        :11333
T14.ROTATE.1:
U 303C, A388,15 :11334      ROLC WR[0],Q ;ROTATE LEFT THE QUADWORD CONSITING OF WR[0] & Q
U 303D, 7610,15 :11335      MOV Q TO LS[T8] ;SAVE Q
U 303E, 8870,CC :11336      JSR [INC.OS] ;INCREMENT OS (WR1=OS)
U 303F, B6F7,15 :11337      MOV LS[SHIFT.OS(4-0)] TO WR[2]
U 3040, 3E89,15 :11338      MOV WR[2] TO LS[ADDRESS.DATA] ;SET "OTHER" FIELD
                        :11339
                        :11340      XOR WR[1] WITH LS[#20] TO Q,
U 3041, CD4A,B5 :11341      DT(LONG)&SET.ALU.CC ;OS = 32 ?
U 3042, 8B04,71 :11342      JMP.IF[NEQ] TO [T14.CCHK1.1] ;JUMP IF NOT
                        :11343
                        :11344      TST WR[3]
U 3043, 2007,B5 :11345      DT(LONG)&SET.ALU.CC ;WR3 SET ?
U 3044, 0B06,21 :11346      JMP.IF[NEQ] TO [T14.3] ;GO TO NEXT SECTION IF SO
U 3045, E5F8,15 :11347      CLR LS[OS] ;RESET OS
U 3046, B69F,95 :11348      MOV LS[ONES] TO WR[3] ;SET WR3
                        :11349
T14.CCHK1.1:
U 3047, 2007,B5 :11350      TST WR[3]
U 3048, 0B04,B9 :11351      DT(LONG)&SET.ALU.CC ;WR3 SET?
U 3049, B6F6,95 :11352      JMP.IF[EQL] TO [T14.CLWR.1] ;JUMP IF NOT
U 304A, 5B00,1E :11353      MOV LS[SHIFT.OS(4-0)] TO WR[1] ;EXPECTED DATA = PATTERN
                        :11354
                        :11355      SKIP
T14.CLWR.1:
U 304B, 2F82,95 :11356      CLR WR[1] ;EXPECTED DATA = 0
U 304C, 0B69,3C :11357      JSR [CHECK.RESULT] ;CHECK WRO
U 304D, 0B05,84 :11358      JMP [T14.LOE.1] ;LOOP ON ERROR IF ENABLED
U 304E, 8870,5C :11359      JSR [INC.EN] ;ERROR NUMBER = 2
                        :11360
                        :11361      TST WR[3]
U 304F, 2007,B5 :11362      DT(LONG)&SET.ALU.CC ;WR3 SET ?
U 3050, 8B05,31 :11363      JMP.IF[NEQ] TO [T14.CLWR.2] ;JUMP IF SO
U 3051, B6F6,95 :11364      MOV LS[SHIFT.OS(4-0)] TO WR[1] ;EXPECTED DATA = PATTERN
U 3052, 5B00,1E :11365      SKIP
T14.CLWR.2:
U 3053, 2F82,95 :11366      CLR WR[1] ;EXPECTED DATA = 0
  
```



```

U 3054, B610,15 :11365      MOV LS[T8] TO WR[0]          ;MOVE SAVED Q TO RECEIVED DATA
U 3055, 0869,3C :11366      JSR [CHECK.RESULT]        ;CHECK Q
U 3056, 0B05,84 :11367      JMP [T14.LOE.1]          ;LOOP ON ERROR IF ENABLED
U 3057, 8B03,34 :11368      JMP [LOOP.T14.1]
:11369
T14.LOE.1:
U 3058, 36F9,15 :11370      MOV LS[OS] TO WR[2]          ;GET INDEX
U 3059, C52D,15 :11371      BIC LS[FFFFFF00] TO WR[2]      ;CLEAR HIGH BITS
:11372
U 305A, 2005,35 :11373      DT(LONG)&SET.ALU.CC          ;OS = 0 ?
U 305B, 8B06,01 :11374      JMP.IF[NEQ] TO [T14.DOS.1]      ;JUMP IF NOT
U 305C, B613,15 :11375      MOV LS[T9] TO WR[2]
U 305D, BEF9,15 :11376      MOV WR[2] TO LS[OS]          ;OS = 31
U 305E, 2F87,95 :11377      CLR WR[3]                  ;WR3 = 0
U 305F, 5B00,1E :11378      SKIP
:11379
T14.DOS.1:
U 3060, 8871,1C :11380      JSR [DEC.OS]                ;DECREMENT OS
U 3061, 8B03,34 :11381      JMP [LOOP.T14.1]              ;GO BACK AND REDO ERROR
:11382
T14.3:
U 3062, 2F87,95 :11383      CLR WR[3]                  ;RESET WR3
U 3063, E5F8,15 :11384      CLR LS[OS]                  ;RESET INDEX
U 3064, B69E,15 :11385      MOV LS[ONES] TO WR[0]          ;SET ALL BITS IN UPPER
:11386
LOOP.T14.3:
U 3065, B643,15 :11387      MOV LS[#2] TO WR[2]
U 3066, 2045,15 :11388      INC WR[2]
U 3067, 3E83,15 :11389      MOV WR[2] TO LS[ERROR.NUMBER]    ;ERROR # = 3
:11390
U 3068, 2007,B5 :11391      DT(LONG)&SET.ALU.CC          ;WR3 SET?
U 3069, 8B06,D9 :11392      JMP.IF[EQL] TO [T14.SET.Q.3]      ;JUMP IF NOT
U 306A, 5FF6,15 :11393      MCOM LS[SHIFT.OS(4-0)] TO WR[0] ;PUT DATA IN WR0
U 306B, 589E,15 :11394      MOV LS[ONES] TO Q
U 306C, 0B07,04 :11395      JMP [T14.ROTATE.3]            ;GO ROTATE
:11396
T14.SET.Q.3:
U 306D, DFF7,15 :11397      MCOM LS[SHIFT.OS(4-0)] TO WR[2] ;PUT DATA IN Q
U 306E, AAC5,15 :11398      MOV WR[2] TO Q
U 306F, B69E,15 :11399      MOV LS[ONES] TO WR[0]
:11400
T14.ROTATE.3:
U 3070, A388,15 :11401      ROLC WR[0],Q                ;ROTATE LEFT THE QUADWORD CONSITING OF WR[0] & Q
U 3071, 7610,15 :11402      MOV Q TO LS[T8]              ;SAVE Q
U 3072, 8870,CC :11403      JSR [INC.OS]                ;INCREMENT OS (WR1=OS)
U 3073, B6F7,15 :11404      MOV LS[SHIFT.OS(4-0)] TO WR[2]
U 3074, 3E89,15 :11405      MOV WR[2] TO LS[ADDRESS.DATA]    ;SET "OTHER" FIELD
:11406
U 3075, CD4A,B5 :11407      DT(LONG)&SET.ALU.CC          ;OS = 32 ?
U 3076, 8B07,B1 :11408      JMP.IF[NEQ] TO [T14.CHCK1.3]      ;JUMP IF NOT
:11409
U 3077, 2007,B5 :11410      DT(LONG)&SET.ALU.CC          ;WR3 SET ?
U 3078, 0B09,71 :11411      JMP.IF[NEQ] TO [T14.5]          ;GO TO NEXT SECTION IF SO
U 3079, E5F8,15 :11412      CLR LS[OS]                  ;RESET OS
U 307A, B69F,95 :11413      MOV LS[ONES] TO WR[3]          ;SET WR3
:11414
T14.CHCK1.3:
:11415
U 307B, 2007,B5 :11416      DT(LONG)&SET.ALU.CC          ;WR3 SET?
U 307C, 8B07,F9 :11417      JMP.IF[EQL] TO [T14.CLWR.3]      ;JUMP IF NOT
U 307D, DFF6,95 :11418      MCOM LS[SHIFT.OS(4-0)] TO WR[1] ;EXPECTED DATA = PATTERN
U 307E, 5B00,1E :11419      SKIP
    
```

```

:11420 T14.CLWR.3:
U 307F, 369E,95 :11421 MOV LS[ONES] TO WR[1] ;EXPECTED DATA = ONES
U 3080, 0869,3C :11422 JSR [CHECK.RESULT] ;CHECK WRO
U 3081, 0B08,C4 :11423 JMP [T14.LOE.3] ;LOOP ON ERROR IF ENABLED
U 3082, 8870,5C :11424 JSR [INC.EN] ;ERROR NUMBER = 4
:11425 TST WR[3]
U 3083, 2007,B5 :11426 DT(LONG)&SET.ALU.CC ;WR3 SET ?
U 3084, 8B08,71 :11427 JMP.IF[NEQ] TO [T14.CLWR.4] ;JUMP IF SO
U 3085, DFF6,95 :11428 MCOM LS[SHIFT.OS(4-0)] TO WR[1] ;EXPECTED DATA = PATTERN
U 3086, 5B00,1E :11429 SKIP
:11430 T14.CLWR.4:
U 3087, 369E,95 :11431 MOV LS[ONES] TO WR[1] ;EXPECTED DATA = ONES
U 3088, B610,15 :11432 MOV LS[T8] TO WR[0] ;MOVE SAVED Q TO RECEIVED DATA
U 3089, 0869,3C :11433 JSR [CHECK.RESULT] ;CHECK Q
U 308A, 0B08,C4 :11434 JMP [T14.LOE.3] ;LOOP ON ERROR IF ENABLED
U 308B, 8B06,54 :11435 JMP [LOOP.T14.3]
:11436 T14.LOE.3:
U 308C, 36F9,15 :11437 MOV LS[OS] TO WR[2] ;GET INDEX
U 308D, C52D,15 :11438 BIC LS[FFFFFFF0] TO WR[2] ;CLEAR HIGH BITS
:11439 TST WR[2]
U 308E, 2005,35 :11440 DT(LONG)&SET.ALU.CC ;OS = 0 ?
U 308F, 8B09,51 :11441 JMP.IF[NEQ] TO [T14.DOS.3] ;JUMP IF NOT
U 3090, 364B,15 :11442 MOV LS[#20] TO WR[2]
U 3091, 2105,15 :11443 DEC WR[2] ;WR2 = 32-1 = 31
U 3092, BEF9,15 :11444 MOV WR[2] TO LS[OS] ;OS = 31
U 3093, 2F87,95 :11445 CLR WR[3] ;WR3 = 0
U 3094, 5B00,1E :11446 SKIP
:11447 T14.DOS.3:
U 3095, 8871,1C :11448 JSR [DEC.OS] ;DECREMENT OS
U 3096, 8B06,54 :11449 JMP [LOOP.T14.3] ;GO BACK AND REDO ERROR
:11450 T14.5:
U 3097, B69F,95 :11451 MOV LS[ONES] TO WR[3] ;SET WR3
U 3098, B613,15 :11452 MOV LS[T9] TO WR[2]
U 3099, BEF9,15 :11453 MOV WR[2] TO LS[OS] ;OS = 31
:11454 LOOP.T14.5:
U 309A, B645,15 :11455 MOV LS[#4] TO WR[2]
U 309B, 2045,15 :11456 INC WR[2]
U 309C, 3E83,15 :11457 MOV WR[2] TO LS[ERROR.NUMBER] ;ERROR # = 5
:11458 TST WR[3]
U 309D, 2007,B5 :11459 DT(LONG)&SET.ALU.CC ;WR3 SET?
U 309E, 8B0A,29 :11460 JMP.IF[EQL] TO [T14.SET.Q.5] ;JUMP IF NOT
U 309F, 36F6,15 :11461 MOV LS[SHIFT.OS(4-0)] TO WR[0] ;PUT DAT IN WRO
U 30A0, AB80,15 :11462 CLR Q
U 30A1, 0B0A,44 :11463 JMP [T14.ROTATE.5] ;GO ROTATE
:11464 T14.SET.Q.5:
U 30A2, D8F6,15 :11465 MOV LS[SHIFT.OS(4-0)] TO Q ;PUT DATA IN Q
U 30A3, 2F80,15 :11466 CLR WR[0]
:11467 T14.ROTATE.5:
U 30A4, 2308,15 :11468 RORC WR[0],Q ;ROTATE RIGHT THE QUADWORD CONSISTING OF WRO & Q
U 30A5, 7610,15 :11469 MOV Q TO LS[T8] ;SAVE Q
U 30A6, 8871,1C :11470 JSR [DEC.OS] ;DECREMENT OS (WR1=OS)
:11471 XOR WR[1] WITH LS[ONES] TO Q,
U 30A7, CD9E,B5 :11472 DT(LONG)&SET.ALU.CC ;OS = -1 ?
U 30A8, 0B0A,E1 :11473 JMP.IF[NEQ] TO [T14.CHCK1.5] ;JUMP IF NOT
:11474 TST WR[3],
    
```



```

U 30A9, 2007,B5 :11475      DT(LONG)&SET.ALU.CC      :WR3 SET?
U 30AA, 8B0C,89 :11476      JMP.IF[EQL] TO [T14.7]      :JUMP IF NOT TO NEXT SECTION
U 30AB, B613,15 :11477      MOV LS[T9] TO WR[2]
U 30AC, BEF9,15 :11478      MOV WR[2] TO LS[OS]      :OS = 31
U 30AD, 2F87,95 :11479      CLR WR[3]      :CLEAR WR3
:11480
T14.CHCK1.5:
:11481      TST WR[3]
:11482      DT(LONG)&SET.ALU.CC      :WR3 SET?
U 30AE, 2007,B5 :11482      JMP.IF[EQL] TO [T14.CLWR.5]      :JUMP IF NOT
U 30AF, 0B0B,29 :11483      MOV LS[SHIFT.OS(4-0)] TO WR[1] :EXPECTED DATA = PATTERN
U 30B0, B6F6,95 :11484      SKIP
:11485
T14.CLWR.5:
:11486      CLR WR[1]      :EXPECTED DATA = 0
U 30B2, 2F82,95 :11487      JSR [CHECK.RESULT]      :CHECK WRO
U 30B3, 0869,3C :11488      JMP [T14.LOE.5]      :LOOP ON ERROR IF ENABLED
U 30B4, 0B0B,F4 :11489      JSR [INC.EN]      :ERROR NUMBER = 6
U 30B5, 8870,5C :11490      TST WR[3]
:11491
U 30B6, 2007,B5 :11492      DT(LONG)&SET.ALU.CC      :WR3 SET ?
U 30B7, 0B0B,A1 :11493      JMP.IF[NEQ] TO [T14.CLWR.6]      :JUMP IF SO
U 30B8, B6F6,95 :11494      MOV LS[SHIFT.OS(4-0)] TO WR[1] :EXPECTED DATA = PATTERN
U 30B9, 5B00,1E :11495      SKIP
:11496
T14.CLWR.6:
:11497      CLR WR[1]      :EXPECTED DATA = 0
U 30BA, 2F82,95 :11497      MOV LS[T8] TO WR[0]      :MOVE SAVED Q TO RECEIVED DATA
U 30BB, B610,15 :11498      JSR [CHECK.RESULT]      :CHECK Q
U 30BC, 0869,3C :11499      JMP [T14.LOE.5]      :LOOP ON ERROR IF ENABLED
U 30BD, 0B0B,F4 :11500      JMP [LOOP.T14.5]
:11501
T14.LOE.5:
:11502      MOV LS[OS] TO WR[2]      :GET INDEX
U 30BF, 36F9,15 :11503      BIC LS[FFFFFF00] TO WR[2]      :CLEAR HIGH BITS
U 30C0, C52D,15 :11504      XOR WR[2] WITH LS[T9] TO Q,
:11505      DT(LONG)&SET.ALU.CC      :OS = 31 ?
U 30C1, 4D13,35 :11506      JMP.IF[NEQ] TO [T14.IOS.5]      :JUMP IF NOT
U 30C2, 8B0C,61 :11507      CLR LS[OS]      :OS = 0
U 30C3, E5F8,15 :11508      MOV LS[ONES] TO WR[3]      :WR3 SET
U 30C4, B69F,95 :11509      SKIP
:11510
T14.IOS.5:
:11511      JSR [INC.OS]      :INCREMENT OS
U 30C6, 8870,CC :11512      JMP [LOOP.T14.5]      :GO BACK AND REDO ERROR
U 30C7, 8B09,A4 :11513
:11514
T14.7:
:11515      MOV LS[ONES] TO WR[3]      :SET WR3
U 30C8, B69F,95 :11515      MOV LS[T9] TO WR[2]
U 30C9, B613,15 :11516      MOV WR[2] TO LS[OS]      :OS = 31
U 30CA, BEF9,15 :11517      MOV LS[ONES] TO Q      :SET ALL BITS IN Q
U 30CB, 589E,15 :11518
:11519
LOOP.T14.7:
:11520      MOV LS[#4] TO WR[2]
U 30CC, B645,15 :11520      INC WR[2]
U 30CD, 2045,15 :11521      BIS LS[#2] TO WR[2]
U 30CE, C743,15 :11522      MOV WR[2] TO LS[ERROR.NUMBER] :ERROR # = 7
U 30CF, 3E83,15 :11523      TST WR[3]
:11524
U 30D0, 2007,B5 :11525      DT(LONG)&SET.ALU.CC      :WR3 SET?
U 30D1, 8B0D,59 :11526      JMP.IF[EQL] TO [T14.SET.Q.7]      :JUMP IF NOT
U 30D2, 5FF6,15 :11527      MCOM LS[SHIFT.OS(4-0)] TO WR[0] :PUT DAT IN WRO
U 30D3, 589E,15 :11528      MOV LS[ONES] TO Q
U 30D4, 8B0D,84 :11529      JMP [T14.ROTATE.7]      :GO ROTATE

```

```

:11530 T14.SET.Q.7:
U 30D5, DFF7.15 :11531 MCOM LS[SHIFT.OS(4-0)] TO WR[2]
U 30D6, AAC5.15 :11532 MOV WR[2] TO Q ;PUT DATA IN Q
U 30D7, B69E.15 :11533 MOV LS[ONES] TO WR[0]
:11534 T14.ROTATE.7:
U 30D8, 2308.15 :11535 RORC WR[0].Q ;ROTATE RIGHT THE QUADWORD CONSIISTING OF WRO & Q
U 30D9, 7610.15 :11536 MOV Q TO LS[T8] ;SAVE Q
U 30DA, 8871.1C :11537 JSR [DEC.OS] ;DECREMENT OS (WR1=OS)
:11538
U 30DB, CD9E.B5 :11539 XOR WR[1] WITH LS[ONES] TO Q,
U 30DC, 8B0E.21 :11540 DT(LONG)&SET.ALU.CC ;OS = -1 ?
:11541 JMP.IF[NEQ] TO [T14.CHCK1.7] ;JUMP IF NOT
U 30DD, 2007.B5 :11542 TST WR[3]
U 30DE, 0B0F.C9 :11543 DT(LONG)&SET.ALU.CC ;WR3 SET?
U 30DF, B613.15 :11544 JMP.IF[EQL] TO [END.T14] ;JUMP IF NOT TO END OF TEST
U 30E0, BEF9.15 :11545 MOV LS[T9] TO WR[2]
U 30E1, 2F87.95 :11546 MOV WR[2] TO LS[OS] ;OS = 31
:11547 CLR WR[3] ;CLEAR WR3
:11548 T14.CHCK1.7:
U 30E2, 2007.B5 :11549 TST WR[3]
U 30E3, 8B0E.69 :11550 DT(LONG)&SET.ALU.CC ;WR3 SET?
U 30E4, DFF6.95 :11551 JMP.IF[EQL] TO [T14.CLWR.7] ;JUMP IF NOT
U 30E5, 5B00.1E :11552 MCOM LS[SHIFT.OS(4-0)] TO WR[1] ;EXPECTED DATA = PATTERN
:11553 SKIP
:11554 T14.CLWR.7:
U 30E6, 369E.95 :11555 MOV LS[ONES] TO WR[1] ;EXPECTED DATA = ONES
U 30E7, 0869.3C :11556 JSR [CHECK.RESULT] ;CHECK WRO
U 30E8, 8B0F.34 :11557 JMP [T14.LOE.7] ;LOOP ON ERROR IF ENABLED
U 30E9, 8870.5C :11558 JSR [INC.EN] ;ERROR NUMBER = 8
:11559 TST WR[3]
U 30EA, 2007.B5 :11560 DT(LONG)&SET.ALU.CC ;WR3 SET ?
U 30EB, 8B0E.E1 :11561 JMP.IF[NEQ] TO [T14.CLWR.8] ;JUMP IF SO
U 30EC, DFF6.95 :11562 MCOM LS[SHIFT.OS(4-0)] TO WR[1] ;EXPECTED DATA = PATTERN
U 30ED, 5B00.1E :11563 SKIP
:11564 T14.CLWR.8:
U 30EE, 369E.95 :11565 MOV LS[ONES] TO WR[1] ;EXPECTED DATA = ONES
U 30EF, B610.15 :11566 MOV LS[T8] TO WR[0] ;MOVE SAVED Q TO RECEIVED DATA
U 30F0, 0869.3C :11567 JSR [CHECK.RESULT] ;CHECK Q
U 30F1, 8B0F.34 :11568 JMP [T14.LOE.7] ;LOOP ON ERROR IF ENABLED
U 30F2, 8B0C.C4 :11569 JMP [LOOP.T14.7]
:11570 T14.LOE.7:
U 30F3, 36F9.15 :11571 MOV LS[OS] TO WR[2] ;GET INDEX
U 30F4, C52D.15 :11572 BIC LS[FFFFFFF0] TO WR[2] ;CLEAR HIGH BITS
:11573 XOR WR[2] WITH LS[T9] TO Q,
U 30F5, 4D13.35 :11574 DT(LONG)&SET.ALU.CC ;OS = 31 ?
U 30F6, 8B0F.A1 :11575 JMP.IF[NEQ] TO [T14.IOS.7] ;JUMP IF NOT
U 30F7, E5F8.15 :11576 CLR LS[OS] ;OS = 0
U 30F8, B69F.95 :11577 MOV LS[ONES] TO WR[3] ;WR3 SET
U 30F9, 5B00.1E :11578 SKIP
:11579 T14.IOS.7:
U 30FA, 8870.CC :11580 JSR [INC.OS] ;INCREMENT OS
U 30FB, 8B0C.C4 :11581 JMP [LOOP.T14.7] ;GO BACK AND REDO ERROR
:11582 END.T14:
    
```


Page "TEST 15 - BUS D D00,D D01 from DATA TYPE CTL PAL (M8390 CPU Module)."

++

Test Description:

This test checks the outputs BUS D D00 and BUS D D01 from the DATA TYPE CTL PAL. These outputs are used internally to control the SIZE REG outputs. SIZE REG will be used in the next test, so testing the BUS D outputs will help reduce the new logic used in the next test. The BUS D outputs can be read directly through the 2901 data path. Only bits 00 and 01 will be checked. The other bits will be masked off. The data path is as follows. A write to LS[SIZE] asserts LOAD STAT L to the DATA TYPE CTL PAL. This enables the Y BUS (bits 00 and 01) to be latched internally at the BUS D D00 and BUS D D01 outputs. The outputs are tri-state and are not enabled at this time. The outputs are enabled when a read from LS[SIZE] is executed by asserting the signal EN STAT REG L. This signal comes from the REG CONTROL PAL. When the outputs are enabled, they are stored in WR[0] of the 2901 via the D input. Data patterns used are 00, 11, 01, and 10 (all binary). This test also checks that the data is not altered if a write to LS [SIZE] is not executed.

Assumptions:

It is assumed that the previous tests have run successfully.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load WR[1] with zeros and WR[0] with ones. Execute MOVE WR[1] TO LS[SIZE].
- 3) Execute MOV LS[SIZE] TO WR[0] and check result for zeros (bits 00 and 01 only).
- 4) Clear WR[1], load WR[0] with ones and execute MOV WR[0] TO LS[OS]. Execute MOV LS[SIZE] to WR[0] and check that data was not altered (still 00(B)).
- 5) Load WR[1] with ones and execute MOV WR[1] TO LS[SIZE].
- 6) Execute MOV LS[SIZE] TO WR[0] and check result for 11(B).
- 7) Repeat step 4 using complement data. Result should still be 11(B).
- 8) Repeat steps 5 and 6 using data of 01(B) and 10(B).

Errors:

- Error 1 - DATA TYPE CTL PAL failed to output correctly. Data of 00(B).
- Error 2 - BUS D D00/BUS D D01 altered by write to OS.
- Error 3 - DATA TYPE CTL PAL failed to output correctly. Data of 11(B).
- Error 4 - BUS D D00/BUS D D01 altered by write to OS.
- Error 5 - DATA TYPE CTL PAL failed to output correctly. Data of 01(B).
- Error 6 - DATA TYPE CTL PAL failed to output correctly. Data of 10(B).

Debug:

Error 1 - This error indicates a failure with the DATA TYPE CTL PAL or

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

M 4
TEST 15 - BUS D D00 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 15 - BUS D D00,D D01 from DATA TYPE CTL PAL (M8390 CPU Module).
Fiche 2 Frame M4
Sequence 257
Rev 01.00
Page 251

:11637 : its inputs. During the write to the DATA TYPE CTL PAL (the MOV WR[1]
:11638 : TO LS[SIZE]), check the inputs for lows on BUS Y D01 H, BUS Y D00 H,
:11639 : and LOAD STAT L. The CLOCK REGS H input should also be checked to
:11640 : assure that there is not an etch break (look for a float). The output
:11641 : is not enabled at this time. If LOAD STAT L is not asserted low,
:11642 : follow it back to the decoder fed by the REG CONTROL PAL and WRITE REGS
:11643 : L. During the u-word that loads the SIZE REG, SEL input 2 (pin 13)
:11644 : should be high and SEL input 1 (pin 14) should be low. The enable on
:11645 : pin 15 should also be low. The select lines come from the REG CONTROL
:11646 : PAL and the inputs during the u-word that loads the SIZE REG should be
:11647 : as follows: CSR 22 should be low, CSR 21 should be high, CSR 18 should
:11648 : be high, CSR 17 should be high, CSR 16 should be low, CSR 10 should be
:11649 : high, and CSR 09 should be low. If all these inputs are OK, suspect
:11650 : the REG CONTROL PAL. If the inputs to the DATA TYPE CTL PAL are OK,
:11651 : check the outputs during the read u-word (MOV LS[SIZE] TO WR[0]). The
:11652 : BUS D outputs should both be low. If not, check the input EN STAT REG
:11653 : L for a low. If this signal is low, the PAL is probably bad. If EN
:11654 : STAT REG L is high, check it at the output of the REG CONTROL PAL.
:11655 : If incorrect, the inputs should be as follows: CSR 22 should be low,
:11656 : CSR 21 should be high, CSR 18 should be high, CSR 17 should be high,
:11657 : CSR 16 should be low, CSR 10 should be high, and CSR 09 should be low.
:11658 : If all these inputs are OK, suspect the REG CONTROL PAL.
:11659 :

:11660 : Error 2 - This error indicates that the signal LOAD STAT L is being
:11661 : asserted when it shouldn't be, or that the DATA TYPE CTL PAL is faulty.
:11662 : Check LOAD STAT L at the input to the DATA TYPE CTL PAL during the
:11663 : write to OS (MOV WR[0] TO LS[OS]) instruction. If LOAD STAT L is high,
:11664 : the DATA TYPE CTL PAL is probably bad. If LOAD STAT L is low, check it
:11665 : at the output of the decoder fed by the REG CONTROL PAL. LOAD STAT L
:11666 : should be high and LOAD Y TO OS L should be low. If both are low, look
:11667 : for a short between the signals or a bad decoder chip.
:11668 :

:11669 : Error 3 - Same as error 1, except the Y BUS inputs should be high. The
:11670 : most likely suspect is the DATA TYPE CTL PAL.
:11671 :

:11672 : Error 4 - Same as error 2.
:11673 :

:11674 : Error 5 - The most likely suspect is the DATA TYPE CTL PAL.
:11675 :

:11676 : Error 6 - Same as error 5.
:11677 :

:11678 :
:11679 : T.15:

U 30FC, B65E,15 :11680 : MOV LS[BEGIN.TEST] TO WR[0] :SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
U 30FD, 3E80,15 :11681 : MOV WR[0] TO LS[CONTROL.STATUS] :SET BIT15 IN CNTROL AND STATUS WORD
:11682 : : BIT15 INDICATES START OF TEST TO CONSOLE
U 30FE, 10E0,15 :11683 : MISC [SET.CP.ATTN] :ISSUE CPU ATTN TO CONSOLE
:11684 :
U 30FF, 8B0F,F4 :11685 : WAIT.T15.0: :LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 3100, B69E,15 :11686 : JMP [WAIT.T15.0]
U 3101, 4540,15 :11687 : MOV LS[ONES] TO WR[0]
U 3102, C542,15 :11688 : BIC LS[BIT0] TO WR[0]
U 3103, 3E8A,15 :11689 : BIC LS[BIT1] TO WR[0]
U 3104, 65A0,15 :11690 : MOV WR[0] TO LS[ERROR.MASK] :ERROR MASK = #FFFFFFFC
U 3105, 2F80,15 :11691 : CLR LS[PREVIOUS.ERROR] :CLEAR PREVIOUS ERROR NUMBER
:11692 : CLR WR[0]


```

U 3106, 2040.15 :11692      INC WR[0]          ;SET WRO TO 1
U 3107, BE82.15 :11693      MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
U 3108, A3C0.15 :11694      ROL WR[0]          ;SET WRO TO 2
U 3109, 3E8C.15 :11695      MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
                                :11696
U 310A, 2F82.95 :11697      LOOP.T15.1:      CLR WR[1]          ;WRI = 0
U 310B, B69E.15 :11698      MOV LS[ONES] TO WR[0]      ;WRO = #FFFFFFFF
U 310C, 3EFC.95 :11699      MOV WR[1] TO LS[SIZE]
U 310D, 36FC.15 :11700      MOV LS[SIZE] TO WR[0]
U 310E, 0869.3C :11701      JSR [CHECK.RESULT]      ;GET DATA
U 310F, 0B10.A4 :11702      JMP [LOOP.T15.1]          ;CHECK FOR 00B
U 3110, 8870.5C :11703      JSR [INC.EN]              ;INCREMENT ERROR NUMBER TO 2
                                :11704
U 3111, 2F82.95 :11705      LOOP.T15.2:      CLR WR[1]          ;WRI = 0
U 3112, B69E.15 :11706      MOV LS[ONES] TO WR[0]      ;WRO = #FFFFFFFF
U 3113, 3EF8.15 :11707      MOV WR[0] TO LS[OS]
U 3114, 36FC.15 :11708      MOV LS[SIZE] TO WR[0]
U 3115, 0869.3C :11709      JSR [CHECK.RESULT]      ;GET DATA
U 3116, 0B11.14 :11710      JMP [LOOP.T15.2]          ;CHECK FOR 00B
U 3117, 8870.5C :11711      JSR [INC.EN]              ;INCREMENT ERROR NUMBER TO 3
                                :11712
U 3118, 369E.95 :11713      LOOP.T15.3:      MOV LS[ONES] TO WR[1]      ;WRI = #FFFFFFFF
U 3119, 3EFC.95 :11714      MOV WR[1] TO LS[SIZE]
U 311A, 36FC.15 :11715      MOV LS[SIZE] TO WR[0]
U 311B, 0869.3C :11716      JSR [CHECK.RESULT]      ;CHECK FOR 11B
U 311C, 0B11.84 :11717      JMP [LOOP.T15.3]          ;LOOP ON ERROR IF ENABLED
U 311D, 8870.5C :11718      JSR [INC.EN]              ;INCREMENT ERROR NUMBER TO 4
                                :11719
U 311E, 2F80.15 :11720      LOOP.T15.4:      CLR WR[0]          ;WRO = 0
U 311F, 369E.95 :11721      MOV LS[ONES] TO WR[1]      ;WRI = #FFFFFFFF
U 3120, 3EF8.15 :11722      MOV WR[0] TO LS[OS]
U 3121, 36FC.15 :11723      MOV LS[SIZE] TO WR[0]
U 3122, 0869.3C :11724      JSR [CHECK.RESULT]      ;GET DATA
U 3123, 0B11.E4 :11725      JMP [LOOP.T15.4]          ;CHECK FOR 11B
U 3124, 8870.5C :11726      JSR [INC.EN]              ;INCREMENT ERROR NUMBER TO 5
                                :11727
U 3125, 3640.95 :11728      LOOP.T15.5:      MOV LS[BIT0] TO WR[1]      ;WRI = #00000001
U 3126, 3EFC.95 :11729      MOV WR[1] TO LS[SIZE]
U 3127, 36FC.15 :11730      MOV LS[SIZE] TO WR[0]
U 3128, 0869.3C :11731      JSR [CHECK.RESULT]      ;CHECK FOR 01B
U 3129, 8B12.54 :11732      JMP [LOOP.T15.5]          ;LOOP ON ERROR IF ENABLED
U 312A, 8870.5C :11733      JSR [INC.EN]              ;INCREMENT ERROR NUMBER TO 6
                                :11734
U 312B, B642.95 :11735      LOOP.T15.6:      MOV LS[BIT1] TO WR[1]      ;WRI = #00000002
U 312C, 3EFC.95 :11736      MOV WR[1] TO LS[SIZE]
U 312D, 36FC.15 :11737      MOV LS[SIZE] TO WR[0]
U 312E, 0869.3C :11738      JSR [CHECK.RESULT]      ;CHECK FOR 10B
U 312F, 0B12.B4 :11739      JMP [LOOP.T15.6]          ;LOOP ON ERROR IF ENABLED
                                :11740
END.T15:
    
```

:11741
:11742
:11743
:11744
:11745
:11746
:11747
:11748
:11749
:11750
:11751
:11752
:11753
:11754
:11755
:11756
:11757
:11758
:11759
:11760
:11761
:11762
:11763
:11764
:11765
:11766
:11767
:11768
:11769
:11770
:11771
:11772
:11773
:11774
:11775
:11776
:11777
:11778
:11779
:11780
:11781
:11782
:11783
:11784
:11785
:11786
:11787
:11788
:11789
:11790
:11791
:11792
:11793
:11794
:11795

Page "TEST 16 - ALU N,Z Byte, Word, Long Tests (M8390 CPU Module)"

++

Test Description:

This test checks the ALU N and ALU Z condition codes for all three data types: byte, word, and longword. The ALU Z bit has been tested previously via the Skip on ALU Z, but only for the longword data type. Initially the CC field in the u-word will specify a longword data type. After this check, the size reg will be loaded and used to control the data type for all three types. The data path is as follows: WR 2 is loaded from LS with a data pattern with SET ALU CC's enabled. Then the ALU N and ALU Z bits are enabled onto the D bus and written into WR 0. WR 1 is written with expected data from LS and the 2 condition code bits are checked. These bits come from D bus bits 2 and 3. All other bits will be masked off and not checked. The data patterns used were picked to provide for three conditions: All bits clear except the N bit for the data type being tested, all bits set except for the N bit for the current data type, and all zeros except for the bits above the current data type (for byte and word). These patterns will assure that the data type selected is enabling the correct 2901 status bits.

Assumptions:

It is assumed that the previous 2901 tests have been run successfully. It is also assumed that the data path from the ALU N,Z PAL to the D BUS has been successfully tested by the console based tests.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load WR 2 with 80000000(H) and longword data type (CC field in u-word=1) and check for Z clear and N set.
- 3) Load WR 2 with 7FFFFFFF(H) and longword data type (CC field in u-word=1) and check for Z clear and N clear.
- 4) Load WR 2 with all 0's and longword data type (CC field in u-word=1) and check for Z set and N clear.
- 5) Load the SIZE REG with 11(B) and repeat steps 2 through 4 using Size Reg to indicate longword (CC field in u-word=2) and check for correct Z and N condition codes.
- 6) Load the SIZE REG with 01(B) and repeat steps 2 through 4 using Size Reg to indicate word (CC field in u-word=2) and check for correct Z and N condition codes. Data used this time is 8000(H), FFFF7FFF(H), and FFFF0000(H).
- 7) Load the SIZE REG with 00(B) and repeat steps 2 through 4 using Size Reg to indicate byte (CC field in u-word=2) and check for correct Z and N condition codes. Data used this time is 80(H), FFFF7F(H), and FFFF00(H).
- 8) Load SIZE REG with 00(B) and set CC field using data of zero. Check that Z bit is set.
- 9) Using a shifting one pattern and check that z bit is clear until the bit moves above the first byte.
- 10) Repeat step 9 for WORD data checking that Z bit remains clear un-

:11796
:11797
:11798
:11799
:11800
:11801
:11802
:11803
:11804
:11805
:11806
:11807
:11808
:11809
:11810
:11811
:11812
:11813
:11814
:11815
:11816
:11817
:11818
:11819
:11820
:11821
:11822
:11823
:11824
:11825
:11826
:11827
:11828
:11829
:11830
:11831
:11832
:11833
:11834
:11835
:11836
:11837
:11838
:11839
:11840
:11841
:11842
:11843
:11844
:11845
:11846
:11847
:11848
:11849
:11850

til the set bit moves above the first word.
11) Repeat step 9 for LONG WORD data checking that Z bit is never set.

Errors:

- Error 1 - ALU Z or ALU N bits failed with negative longword data. Data type determined by CC field. Z bit is bit 2, N is bit 3.
- Error 2 - ALU Z or ALU N bits failed with non-negative and non-zero longword data. Data type determined by CC field. Z bit is bit 2, N is bit 3.
- Error 3 - ALU Z or ALU N bits failed with zero longword data. Data type determined by CC field. Z bit is bit 2, N is bit 3.
- Error 4 - ALU Z or ALU N bits failed with negative longword data. Data type determined by SIZE REG. Z bit is bit 2, N is bit 3.
- Error 5 - ALU Z or ALU N bits failed with non-negative and non-zero longword data. Data type determined by SIZE REG. Z bit is bit 2, N is bit 3.
- Error 6 - ALU Z or ALU N bits failed with zero longword data. Data type determined by SIZE REG. Z bit is bit 2, N is bit 3.
- Error 7 - ALU Z or ALU N bits failed with negative word data. Data type determined by SIZE REG. Z bit is bit 2, N is bit 3.
- Error 8 - ALU Z or ALU N bits failed with non-negative and non-zero word data. Data type determined by SIZE REG. Z bit is bit 2, N is bit 3.
- Error 9 - ALU Z or ALU N bits failed with zero word data. Data type determined by SIZE REG. Z bit is bit 2, N is bit 3.
- Error A - ALU Z or ALU N bits failed with negative byte data. Data type determined by SIZE REG. Z bit is bit 2, N is bit 3.
- Error B - ALU Z or ALU N bits failed with non-negative and non-zero byte data. Data type determined by SIZE REG. Z bit is bit 2, N is bit 3.
- Error C - ALU Z or ALU N bits failed with zero byte data. Data type determined by SIZE REG. Z bit is bit 2, N is bit 3.
- Error D - ALU Z failed with data of zero, size = byte.
- Error E - ALU Z failed with data of shifting ones, size = byte.
- Error F - ALU Z failed with data of shifting ones, size = word.
- Error 10 - ALU Z failed with data of shifting ones, size = long word.

Debug:

Error 1 - This error could be caused by a number of problems. First check the Z LONG H or N LONG H going into the ALU N,Z PAL. N should be high and Z should be low during the u-word that sets the ALU codes. If these are incorrect, then check the output from the 2901. Suspect a bad 2901 if the error is found there. If Z and N into the pal are OK, then check the inputs LOAD ALU CC L for a high, ALU CC F0 H for a low, and ALU CC F1 H for a high. If these signals are OK then the ALU N,Z PAL is probably bad (the rest of the data path was tested previously by the console). The signals ALU CC F0 H and ALU CC F1 H come from the CC CONTROL PAL. If these signals are bad, check CSR 22 for a low, CSR 21 for a high, CSR 6 for a low, and CSR 5 for a high.

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 16 - ALU N,Z B 3-MAR-82 D 5
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 16 - ALU N,Z Byte, Word, Long Tests (M8390 CPU Module) Fiche 2 Frame D5
Sequence 261 Page 255
Rev 01.00

:11851 : If these signals are OK, but F0 or F1 are not, suspect a bad CC CONTROL
:11852 : PAL. LOAD ALU CC L comes from the decoder fed by the REG CONTROL PAL
:11853 : and the signal WRITE REGS L. WRITE REGS L should be high, thus dis-
:11854 : abling the decoder and producing the high needed on LOAD ALU CC L.
:11855 :
:11856 : Error 2 - Same as error 1 except Z should be low and N should be low
:11857 : going into the ALU N,Z PAL. Also the input to the CC CONTROL PAL
:11858 : on CSR 22 should be high and CSR 21 is a don't care.
:11859 :
:11860 : Error 3 - Same as error 1 except Z should be high and N should be low.
:11861 : The other signals are identical.
:11862 :
:11863 : Error 4 - This error indicates a problem with the SIZE REG or CC
:11864 : CONTROL PAL. At the output of the CC CONTROL PAL, ALU CC F0 H should
:11865 : be low and ALU CC F1 H should be high during the u-word that sets the
:11866 : ALU codes. The logic after this was used in the previous errors. The
:11867 : inputs to the CC CONTROL PAL should be as follows: CSR 22 should be
:11868 : low, CSR 21 should be high, CSR 6 should be high, CSR 5 should be low,
:11869 : and SIZE REG 1 H should be high. If these inputs are OK, the PAL
:11870 : itself is suspect. The SIZE REG 1 input comes from the DATA TYPE CTL
:11871 : PAL. If the output is bad at this PAL, suspect the DATA TYPE CTL PAL.
:11872 : (note: The SIZE REG outputs are driven internally by BUS D D00 and
:11873 : BUS D D01, which don't become valid until the end of the u-cycle that
:11874 : loads the SIZE REG. These outputs were checked in the previous test,
:11875 : so the inputs are known to be good).
:11876 :
:11877 : Error 5 - Same as Error 4 except for the following inputs: CC CONTROL
:11878 : PAL - CSR 22 should be high, CSR 21 is a don't care
:11879 :
:11880 : Error 6 - Same as error 4.
:11881 :
:11882 : Error 7 - This error indicates a problem with the 2901, or one of sev-
:11883 : eral PALS. First check the Z WORD H and N WORD H inputs to the ALU
:11884 : N,Z PAL during the u-word that sets the ALU codes. Z should be low
:11885 : and N should be high. Check the outputs at the 2901 if these signals
:11886 : are bad. If the output is bad at the 2901, suspect the 2901 itself.
:11887 : If N WORD and Z WORD are OK, check ALU CC F0 H for a high and ALU CC
:11888 : F1 H for a low at the ALU N,Z PAL. If these inputs are OK, then
:11889 : suspect the ALU N,Z PAL. F0 and F1 come from the CC CONTROL PAL.
:11890 : If they are incorrect here then check the inputs to the CC CONTROL
:11891 : PAL for the following: CSR 22 should be low, CSR 21 should be high,
:11892 : CSR 06 should be high, CSR 05 should be low, SIZE REG 1 should be low,
:11893 : and SIZE REG 0 should be high. If these inputs are OK, suspect the
:11894 : CC CONTROL PAL. The SIZE REG inputs come from the DATA TYPE CTL PAL.
:11895 : If these signals were incorrect, check them at the output of this PAL.
:11896 : If incorrect, suspect the DATA TYPE CTL PAL itself.
:11897 :
:11898 : Error 8 - Same as error 7 except for the following inputs: ALU N,Z
:11899 : PAL - N WORD H should be low. CC CONTROL PAL - CSR 22 should be high
:11900 : and CSR 21 is a don't care. All other signals are identical to error
:11901 : 7.
:11902 :
:11903 : Error 9 - Same as error 7 except N WORD H should be low and Z WORD H
:11904 : should be high at the ALU N,Z PAL.
:11905 :

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 16 - ALU N,Z B 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 16 - ALU N,Z Byte, Word, Long Tests (M8390 CPU Module)

Fiche 2 Frame E5

Sequence 262
Rev 01.00 Page 256

```
:11906 : Error A - This error indicates a problem with the 2901, or one of sev-
:11907 : eral PALS. First check the Z BYTE H and N BYTE H inputs to the ALU
:11908 : N.Z PAL during the u-word that sets the ALU codes. Z should be low
:11909 : and N should be high. Check the outputs at the 2901 if these signals
:11910 : are bad. If the output is bad at the 2901, suspect the 2901 itself.
:11911 : If N BYTE and Z BYTE are OK, check ALU CC F0 H for a low and ALU CC
:11912 : F1 H for a low at the ALU N.Z PAL. If these inputs are OK, then
:11913 : suspect the ALU N.Z PAL. F0 and F1 come from the CC CONTROL PAL.
:11914 : If they are incorrect here then check the inputs to the CC CONTROL
:11915 : PAL for the following: CSR 22 should be low, CSR 21 should be high,
:11916 : CSR 06 should be high, CSR 05 should be low, SIZE REG 1 should be low,
:11917 : and SIZE REG 0 should be low. If these inputs are OK, suspect the
:11918 : CC CONTROL PAL. The SIZE REG inputs come from the DATA TYPE CTL PAL.
:11919 : If these signals were incorrect, check them at the output of this PAL.
:11920 : If incorrect, suspect the DATA TYPE CONTROL PAL itself.
:11921 :
:11922 : Error B - Same as error A except for the following inputs: ALU N.Z
:11923 : PAL - N BYTE H should be low. CC CONTROL PAL - CSR 22 should be high
:11924 : and CSR 21 is a don't care. All other signals are identical to error
:11925 : A.
:11926 :
:11927 : Error C - Same as error A except N BYTE H should be low and Z BYTE H
:11928 : should be high. All other signals are identical to error A.
:11929 :
:11930 :
:11931 : T.16:
U 3130, B65E,15 :11932 MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
U 3131, 3E80,15 :11933 MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
:11934 : BIT15 INDICATES START OF TEST TO CONSOLE
U 3132, 10E0,15 :11935 MISC [SET.CP.ATTN] ;ISSUE CPU ATTN TO CONSOLE
:11936 :
WAIT.T16.0: :11937 JMP [WAIT.T16.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
:11938 :
U 3133, 0B13,34 :11937 MOV LS[ONES] TO WR[0]
U 3134, B69E,15 :11938 BIC LS[BIT2] TO WR[0]
U 3135, C544,15 :11939 BIC LS[BIT3] TO WR[0]
U 3136, 4546,15 :11940 MOV WR[0] TO LS[ERROR.MASK] ;ERROR MASK = #FFFFFFF3
U 3137, 3E8A,15 :11941 CLR LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
U 3138, 65A0,15 :11942 CLR WR[0]
U 3139, 2F80,15 :11943 INC WR[0] ;SET WRO TO 1
U 313A, 2040,15 :11944 MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
U 313B, BE82,15 :11945 POL WR[0] ;SET WRO TO 2
U 313C, A3C0,15 :11946 MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
U 313D, 3E8C,15 :11947
:11948 :
:11949 : LOOP.T16.1:
:11950 : MOV LS[BIT31] TO WR[2],
:11951 : DT(LONG)&SET.ALU.CC ;WR2 = #80000000
U 313E, 367F,35 :11951 MOV LS[ALU.CC] TO WR[0] ;MOVE CC'S TO WRO
U 313F, B7FC,15 :11952 MOV LS[BIT3] TO WR[1] ;EXPECTED DATA = BIT3 SET
U 3140, 3646,95 :11953 JSR [CHECK.RESULT] ;CHECK FOR N SET, Z CLEAR
U 3141, 0869,3C :11954 JMP [LOOP.T16.1] ;LOOP ON ERROR IF ENABLED
U 3142, 8B13,E4 :11955 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
U 3143, 8870,5C :11956
:11957 :
:11958 : LOOP.T16.2:
:11959 : MCOM LS[BIT31] TO WR[2],
:11960 : DT(LONG)&SET.ALU.CC ;WR2 = #7FFFFFFF
U 3144, 5F7F,35 :11959 MOV LS[ALU.CC] TO WR[0] ;MOVE CC'S TO WR[0]
U 3145, B7FC,15 :11960
```

U 3146, 2F82.95	:11961	CLR WR[1]	:EXPECTED DATA = CLEAR
U 3147, 0869.3C	:11962	JSR [CHECK.RESULT]	:CHECK FOR N AND Z CLEAR
U 3148, 0B14.44	:11963	JMP [LOOP.T16.2]	:LOOP ON ERROR IF ENABLED
U 3149, 8870.5C	:11964	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 3
	:11965	LOOP.T16.3:	
	:11966	MOV LS[ZERO] TO WR[2],	
U 314A, 369D.35	:11967	DT(LONG)&SET.ALU.CC	:WR2 = #00000000
U 314B, B7FC.15	:11968	MOV LS[ALU.CC] TO WR[0]	:MOVE CC'S TO WR[0]
U 314C, B644.95	:11969	MOV LS[BIT2] TO WR[1]	:EXPECTED DATA = BIT2 SET
U 314D, 0869.3C	:11970	JSR [CHECK.RESULT]	:CHECK FOR N CLEAR, Z SET
U 314E, 8B14.A4	:11971	JMP [LOOP.T16.3]	:LOOP ON ERROR IF ENABLED
U 314F, 8870.5C	:11972	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 4
U 3150, 3641.15	:11973	MOV LS[BIT0] TO WR[2]	
U 3151, C743.15	:11974	BIS LS[BIT1] TO WR[2]	:WR2 = #00000003 (11B)
U 3152, 3EFD.15	:11975	MOV WR[2] TO LS[SIZE]	:SET SIZE TO LONGWORD
	:11976	LOOP.T16.4:	
	:11977	MOV LS[BIT31] TO WR[2],	
U 3153, 367F.55	:11978	DT(SIZE)&SET.ALU.CC	:WR2 = #80000000
U 3154, B7FC.15	:11979	MOV LS[ALU.CC] TO WR[0]	:MOVE CC'S TO WR0
U 3155, 3646.95	:11980	MOV LS[BIT3] TO WR[1]	:EXPECTED DATA = BIT3 SET
U 3156, 0869.3C	:11981	JSR [CHECK.RESULT]	:CHECK FOR N SET, Z CLEAR
U 3157, 0B15.34	:11982	JMP [LOOP.T16.4]	:LOOP ON ERROR IF ENABLED
U 3158, 8870.5C	:11983	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 5
	:11984	LOOP.T16.5:	
	:11985	MCOM LS[BIT31] TO WR[2],	
U 3159, 5F7F.55	:11986	DT(SIZE)&SET.ALU.CC	:WR2 = #7FFFFFFF
U 315A, B7FC.15	:11987	MOV LS[ALU.CC] TO WR[0]	:MOVE CC'S TO WR[0]
U 315B, 2F82.95	:11988	CLR WR[1]	:EXPECTED DATA = CLEAR
U 315C, 0869.3C	:11989	JSR [CHECK.RESULT]	:CHECK FOR N AND Z CLEAR
U 315D, 0B15.94	:11990	JMP [LOOP.T16.5]	:LOOP ON ERROR IF ENABLED
U 315E, 8870.5C	:11991	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 6
	:11992	LOOP.T16.6:	
	:11993	MOV LS[ZERO] TO WR[2],	
U 315F, 369D.55	:11994	DT(SIZE)&SET.ALU.CC	:WR2 = #00000000
U 3160, B7FC.15	:11995	MOV LS[ALU.CC] TO WR[0]	:MOVE CC'S TO WR[0]
U 3161, B644.95	:11996	MOV LS[BIT2] TO WR[1]	:EXPECTED DATA = BIT2 SET
U 3162, 0869.3C	:11997	JSR [CHECK.RESULT]	:CHECK FOR N CLEAR, Z SET
U 3163, 0B15.F4	:11998	JMP [LOOP.T16.6]	:LOOP ON ERROR IF ENABLED
U 3164, 8870.5C	:11999	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 7
U 3165, 3641.15	:12000	MOV LS[BIT0] TO WR[2]	:WR2 = #00000001 (01B)
U 3166, 3EFD.15	:12001	MOV WR[2] TO LS[SIZE]	:SET SIZE TO WORD
U 3167, DF29.15	:12002	MCOM LS[FFFF] TO WR[2]	
U 3168, BE11.15	:12003	MOV WR[2] TO LS[8]	:PATTERN FOR Z BIT WORD TEST = #FFFF0000
	:12004	LOOP.T16.7:	
	:12005	MOV LS[BIT15] TO WR[2],	
U 3169, B65F.55	:12006	DT(SIZE)&SET.ALU.CC	:WR2 = #8000
U 316A, B7FC.15	:12007	MOV LS[ALU.CC] TO WR[0]	:MOVE CC'S TO WR0
U 316B, 3646.95	:12008	MOV LS[BIT3] TO WR[1]	:EXPECTED DATA = BIT3 SET
U 316C, 0869.3C	:12009	JSR [CHECK.RESULT]	:CHECK FOR N SET, Z CLEAR
U 316D, 0B16.94	:12010	JMP [LOOP.T16.7]	:LOOP ON ERROR IF ENABLED
U 316E, 8870.5C	:12011	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 8
	:12012	LOOP.T16.8:	
	:12013	MCOM LS[BIT15] TO WR[2],	
U 316F, DF5F.55	:12014	DT(SIZE)&SET.ALU.CC	:WR2 = #FFFF7FFF
U 3170, B7FC.15	:12015	MOV LS[ALU.CC] TO WR[0]	:MOVE CC'S TO WR[0]


```

U 3171, 2F82,95 :12016 CLR WR[1] ;EXPECTED DATA = CLEAR
U 3172, 0869,3C :12017 JSR [CHECK.RESULT] ;CHECK FOR N AND Z CLEAR
U 3173, 0B16,F4 :12018 JMP [LOOP.T16.8] ;LOOP ON ERROR IF ENABLED
U 3174, 8870,5C :12019 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 9
:12020
LOOP.T16.9:
:12021 MOV LS[BIT8] TO WR[2],
:12022 DT(SIZE)&SET,ALU.CC ;WR2 = #FFFF0000
U 3175, B611,55 :12022 MOV LS[ALU.CC] TO WR[0] ;MOVE CC'S TO WR[0]
U 3176, B7FC,15 :12023 MOV LS[BIT2] TO WR[1] ;EXPECTED DATA = BIT2 SET
U 3177, B644,95 :12024 JSR [CHECK.RESULT] ;CHECK FOR N CLEAR, Z SET
U 3178, 0869,3C :12025 JMP [LOOP.T16.9] ;LOOP ON ERROR IF ENABLED
U 3179, 8B17,54 :12026 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO A
U 317A, 8870,5C :12027 MOV LS[ZERO] TO WR[2] ;WR2 = #00000000 (00B)
U 317B, B69D,15 :12028 MOV WR[2] TO LS[SIZE] ;SET SIZE TO WORD
U 317C, 3EFD,15 :12029
:12030
LOOP.T16.A:
:12031 MOV LS[BIT7] TO WR[2],
:12032 DT(SIZE)&SET,ALU.CC ;WR2 = #80
U 317D, 364F,55 :12032 MOV LS[ALU.CC] TO WR[0] ;MOVE CC'S TO WR[0]
U 317E, B7FC,15 :12033 MOV LS[BIT3] TO WR[1] ;EXPECTED DATA = BIT3 SET
U 317F, 3646,95 :12034 JSR [CHECK.RESULT] ;CHECK FOR N SET, Z CLEAR
U 3180, 0869,3C :12035 JMP [LOOP.T16.A] ;LOOP ON ERROR IF ENABLED
U 3181, 0B17,D4 :12036 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO B
U 3182, 8870,5C :12037
:12038
LOOP.T16.B:
:12039 MCOM LS[BIT7] TO WR[2],
:12040 DT(SIZE)&SET,ALU.CC ;WR2 = #FFFFFF7FFF
U 3183, 5F4F,55 :12040 MOV LS[ALU.CC] TO WR[0] ;MOVE CC'S TO WR[0]
U 3184, B7FC,15 :12041 CLR WR[1] ;EXPECTED DATA = CLEAR
U 3185, 2F82,95 :12042 JSR [CHECK.RESULT] ;CHECK FOR N AND Z CLEAR
U 3186, 0869,3C :12043 JMP [LOOP.T16.B] ;LOOP ON ERROR IF ENABLED
U 3187, 8B18,34 :12044 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO C
U 3188, 8870,5C :12045
:12046
LOOP.T16.C:
:12047 MOV LS[#FFFFFFF00] TO WR[2],
:12048 DT(SIZE)&SET,ALU.CC ;WR2 = #FFFFFFF00
U 3189, B62D,55 :12048 MOV LS[ALU.CC] TO WR[0] ;MOVE CC'S TO WR[0]
U 318A, B7FC,15 :12049 MOV LS[BIT2] TO WR[1] ;EXPECTED DATA = BIT2 SET
U 318B, B644,95 :12050 JSR [CHECK.RESULT] ;CHECK FOR N CLEAR, Z SET
U 318C, 0869,3C :12051 JMP [LOOP.T16.C] ;LOOP ON ERROR IF ENABLED
U 318D, 8B18,94 :12052 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO D
U 318E, 8870,5C :12053 CLR WR[1] ;CLEAR OS
U 318F, 2F82,95 :12054 MOV WR[1] TO LS[OS] ;SIZE = BYTE
U 3190, BEF8,95 :12055 MOV WR[1] TO LS[SIZE] ;MASK = #FFFFFFFB
U 3191, 3EFC,95 :12056 MCOM LS[BIT2] TO WR[0] ;WR1 = 0
U 3192, 5F44,15 :12057 MOV WR[0] TO LS[ERROR.MASK] ;SET CC'S
U 3193, 3E8A,15 :12058
:12059
LOOP.T16.D:
:12060 CLR WR[1] ;GET CC'S
:12061 MOV WR[1] TO WR[0], ;EXPECTED = BIT2 (Z BIT)
:12062 DT(SIZE)&SET,ALU.CC ;CHECK FOR Z SET
U 3195, A002,55 :12062 MOV LS[ALU.CC] TO WR[0] ;LOOP ON ERROR IF ENABLED
U 3196, B7FC,15 :12063 MOV LS[BIT2] TO WR[1] ;INCREMENT ERROR NUMBER TO E
U 3197, B644,95 :12064 JSR [CHECK.RESULT] ;CLEAR FLAG
U 3198, 0869,3C :12065 JMP [LOOP.T16.D]
U 3199, 8B19,44 :12066 JSR [INC.EN]
U 319A, 8870,5C :12067 CLR WR[3]
U 319B, 2F87,95 :12068
:12069
LOOP.T16.E:
:12070 CLR WR[1] ;CLEAR EXPECTED
U 319C, 2F82,95 :12070
  
```

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) TEST 16 - ALU N,Z B 3-MAR-82 H 5
3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module Sequence 265
TEST 16 - ALU N,Z Byte, Word, Long Tests (M8390 CPU Module) Rev 01.00 Page 259

```
:12071      MOV LS[SHIFT.OS(4-0)] TO WR[2],
U 319D, 36F7,55 :12072      DT(SIZE)&SET.ALU.CC      ;SET CC'S ON SHIFTING ONE PATTERN
U 319E, B7FC,15 :12073      MOV LS[ALU.CC] TO WR[0]      ;GET CC'S
:12074      XOR WR[2] WITH LS[BIT8] TO Q,
U 319F, 4D51,35 :12075      DT(LONG)&SET.ALU.CC      ;IS SHIFTING ONE PAST FIRST BYTE?
U 31A0, DB00,01 :12076      SKIP.IF[NEQ]      ;SKIP IF NOT
U 31A1, B69F,95 :12077      MOV LS[ONES] TO WR[3]
:12078      TST WR[3],
U 31A2, 2007,B5 :12079      DT(LONG)&SET.ALU.CC      ;FLAG SET?
U 31A3, 5B00,09 :12080      SKIP.IF[EQL]      ;SKIP IF NOT
U 31A4, B644,95 :12081      MOV LS[BIT2] TO WR[1]      ;EXPECTED = BIT2 (Z BIT)
U 31A5, 0869,3C :12082      JSR [CHECK.RESULT]      ;CHECK FOR APPROPRIATE RESULT
U 31A6, 0B19,C4 :12083      JMP [LOOP.T16.E]      ;LOOP ON ERROR IF ENABLED
:12084      XOR WR[2] WITH LS[BIT31] TO Q,
U 31A7, 4D7F,35 :12085      DT(LONG)&SET.ALU.CC      ;LAST BIT TESTED?
U 31A8, 0B1A,B9 :12086      JMP.IF[EQL] TO [T16.F]      ;JUMP IF SO
U 31A9, 8870,CC :12087      JSR [INC.OS]      ;INCREMENT THE OS REGISTER
U 31AA, 0B19,C4 :12088      JMP [LOOP.T16.E]      ;GO DO NEXT BIT
:12089
T16.F:      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO F
:12090      CLR WR[3]      ;CLEAR FLAG
U 31AB, 8870,5C :12091      CLR WR[3]      ;CLEAR OS
U 31AC, 2F87,95 :12092      MOV WR[3] TO LS[OS]
U 31AD, 3EF9,95 :12093      MOV LS[BIT0] TO WR[1]
U 31AE, 3640,95 :12094      MOV WR[1] TO LS[SIZE]      ;SIZE = WORD
:12095
LOOP.T16.F:      CLR WR[1]      ;CLEAR EXPECTED
:12096      MOV LS[SHIFT.OS(4-0)] TO WR[2],
U 31B0, 2F82,95 :12097      DT(SIZE)&SET.ALU.CC      ;SET CC'S ON SHIFTING ONE PATTERN
U 31B1, 36F7,55 :12098      MOV LS[ALU.CC] TO WR[0]      ;GET CC'S
U 31B2, B7FC,15 :12099      XOR WR[2] WITH LS[BIT16] TO Q,
:12100      DT(LONG)&SET.ALU.CC      ;IS SHIFTING ONE PAST FIRST WORD?
U 31B3, 4D61,35 :12101      SKIP.IF[NEQ]      ;SKIP IF NOT
U 31B4, DB00,01 :12102      MOV LS[ONES] TO WR[3]
:12103      TST WR[3],
U 31B5, B69F,95 :12104      DT(LONG)&SET.ALU.CC      ;FLAG SET?
U 31B6, 2007,B5 :12105      SKIP.IF[EQL]      ;SKIP IF NOT
U 31B7, 5B00,09 :12106      MOV LS[BIT2] TO WR[1]      ;EXPECTED = BIT2 (Z BIT)
U 31B8, B644,95 :12107      JSR [CHECK.RESULT]      ;CHECK FOR APPROPRIATE RESULT
U 31B9, 0869,3C :12108      JMP [LOOP.T16.F]      ;LOOP ON ERROR IF ENABLED
U 31BA, 8B1B,04 :12109      XOR WR[2] WITH LS[BIT31] TO Q,
:12110      DT(LONG)&SET.ALU.CC      ;LAST BIT TESTED?
U 31BB, 4D7F,35 :12111      JMP.IF[EQL] TO [T16.10]      ;JUMP IF SO
U 31BC, 0B1B,F9 :12112      JSR [INC.OS]      ;INCREMENT THE OS REGISTER
U 31BD, 8870,CC :12113      JMP [LOOP.T16.F]      ;GO DO NEXT BIT
:12114
T16.10:      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO F
:12115      CLR WR[3]      ;CLEAR FLAG
U 31BF, 8870,5C :12116      CLR WR[3]      ;CLEAR OS
U 31C0, 2F87,95 :12117      MOV WR[3] TO LS[OS]
U 31C1, 3EF9,95 :12118      MOV LS[SHIFT.OS(4-0)] TO WR[2],
:12119      DT(LONG)&SET.ALU.CC      ;SET CC'S ON SHIFTING ONE PATTERN
U 31C2, 36F7,35 :12120      MOV LS[ALU.CC] TO WR[0]      ;GET CC'S
U 31C3, B7FC,15 :12121      CLR WR[1]      ;EXPECTED = 0
U 31C4, 2F82,95 :12122      JSR [CHECK.RESULT]      ;CHECK FOR Z BIT ALWAYS CLEAR
U 31C5, 0869,3C :12123      JMP [LOOP.T16.10]      ;LOOP ON ERROR IF ENABLED
U 31C6, 8B1C,24 :12124
```


U 31C7, 4D7F,35	:12126	XOR WR[2] WITH LS[BIT31] TO Q,	
U 31C8, 0B1C,B9	:12127	DT(LONG)&SET.ALU.CC	:LAST BIT TESTED?
U 31C9, 8870,CC	:12128	JMP.[IF[EQL] TO [END.T16]	:JUMP IF SO
U 31CA, 8B1C,24	:12129	JSR [INC.OS]	:INCREMENT THE OS REGISTER
	:12130	JMP [LOOP.T16.10]	:GO DO NEXT BIT
	:12131	END.T16:	

:12132
:12133
:12134
:12135
:12136
:12137
:12138
:12139
:12140
:12141
:12142
:12143
:12144
:12145
:12146
:12147
:12148
:12149
:12150
:12151
:12152
:12153
:12154
:12155
:12156
:12157
:12158
:12159
:12160
:12161
:12162
:12163
:12164
:12165
:12166
:12167
:12168
:12169
:12170
:12171
:12172
:12173
:12174
:12175
:12176
:12177
:12178
:12179
:12180
:12181
:12182
:12183
:12184
:12185
:12186

Page "TEST 17 - ALU V,C Byte, Word, Long Tests (M8390 CPU Module)"

++

Test Description:

This test checks the ALU V and ALU C condition codes for all three data types; byte, word, and longword. The size reg will be loaded and used to control the data type for all three types. The data path is as follows: WR 2 is loaded from LS with a data pattern and an ADD LS TO WR is executed with SET ALU CC's enabled. Then the ALU V and ALU C bits are enabled onto the D bus and written into WR 0. WR 1 is written with expected data from LS and the 2 condition code bits are checked. These bits come from D bus bits 0 and 1. All other bits will be masked off and not checked. The data patterns used are all ones and shifting ones, all ones and all zeros, and other combinations necessary to fully test the ALU V and ALU C bits. These patterns will also assure that the data type selected is enabling the correct 2901 status bits.

Assumptions:

It is assumed that the previous 2901 tests and the previous ALU Z and ALU N tests have been run successfully. It is also assumed that the data path from the ALU V,C PAL to the D BUS has been successfully tested by the console based tests.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load SIZE REG with 11(B) to indicate longword data type.
- 3) Load WR 2 with all ones and then execute ADD LS TO WR with data of shifting 1's from LS starting at bit 27. Also enable loading of ALU CC's in this instruction with CC field of u-word=2.
- 4) Load WR 1 with expected data, load WR 0 with ALU C and ALU V and check for ALU C set and ALU V clear.
- 5) Repeat steps 3 and 4, shifting the 1 left, until an add with the 1 in bit 30 has been tested.
- 6) Shift the 1 bit to bit 31 and repeat step 3. Load WR 1 with expected data, load WR 0 with ALU C and ALU V, and check for ALU C set and ALU V set.
- 7) Set up data pattern of 7FFFFFFF in WR 2 and then execute ADD LS TO WR with shifting 1's from LS starting at bit 27. Also enable loading of ALU CC's in the instruction with CC field of u-word=2.
- 8) Load WR 1 with expected data, move ALU C and ALU V to WR 0, and check result for C clear and V set.
- 9) Repeat steps 7 and 8, shifting the 1 left, until an add with the 1 in bit 30 has been tested.
- 10) Shift the 1 bit to bit 31 and repeat step 8. Load WR 1 with expected data, load WR 0 with ALU C and ALU V, and check for ALU C clear and ALU V clear.
- 11) Load SIZE REG with 01(B) to indicate word data type.
- 12) Repeat steps 3 through 10 above with data of FFFF for step 3 and data of 7FFF for step 7. Substitute bit 11 for bit 27, bit 14 for bit 30, and bit 15 for bit 31 above.

:12187 : 13) Load SIZE REG with 00(B) to indicate byte data type.
:12188 : 14) Repeat steps 3 through 10 above with data of FF for step 3 and
:12189 : data of 7F for step 7. Substitute bit 3 for bit 27, bit 6 for
:12190 : bit 30, and bit 7 for bit 31 above.
:12191 :
:12192 :
:12193 :
:12194 :
:12195 :
:12196 :
:12197 :
:12198 :
:12199 :
:12200 :
:12201 :
:12202 :
:12203 :
:12204 :
:12205 :
:12206 :
:12207 :
:12208 :
:12209 :
:12210 :
:12211 :
:12212 :
:12213 :
:12214 :
:12215 :
:12216 :
:12217 :
:12218 :
:12219 :
:12220 :
:12221 :
:12222 :
:12223 :
:12224 :
:12225 :
:12226 :
:12227 :
:12228 :
:12229 :
:12230 :
:12231 :
:12232 :
:12233 :
:12234 :
:12235 :
:12236 :
:12237 :
:12238 :
:12239 :
:12240 :
:12241 :

Errors:

- Error 1 - ALU C and/or ALU V failed. ALU C should have set, ALU V should have cleared. Data type=longword.
- Error 2 - ALU C and/or ALU V failed. ALU C should have set, ALU V should have set. Data type=longword.
- Error 3 - ALU C and/or ALU V failed. ALU C should have cleared, ALU V should have set. Data type=longword.
- Error 4 - ALU C and/or ALU V failed. ALU C should have cleared, ALU V should have cleared. Data type=longword.
- Error 5 - ALU C and/or ALU V failed. ALU C should have set, ALU V should have cleared. Data type=word.
- Error 6 - ALU C and/or ALU V failed. ALU C should have set, ALU V should have set. Data type=word.
- Error 7 - ALU C and/or ALU V failed. ALU C should have cleared, ALU V should have set. Data type=word.
- Error 8 - ALU C and/or ALU V failed. ALU C should have cleared, ALU V should have cleared. Data type=word.
- Error 9 - ALU C and/or ALU V failed. ALU C should have set, ALU V should have cleared. Data type=byte.
- Error A - ALU C and/or ALU V failed. ALU C should have set, ALU V should have set. Data type=byte.
- Error B - ALU C and/or ALU V failed. ALU C should have cleared, ALU V should have set. Data type=byte.
- Error C - ALU C and/or ALU V failed. ALU C should have cleared, ALU V should have cleared. Data type=byte.

Debug:

Error 1 - This error could be caused by a number of problems. First check the C LONG H or V LONG H going into the ALU V.C PAL. C should be high and V should be low during the ADD LS TO WR instruction. If these are incorrect, then check the output from the 2901. Suspect a bad 2901 if the error is found there. If C and V into the pal are OK, then check the inputs LOAD ALU CC L for a high, ALU CC FO H for a low, and ALU CC F1 H for a high. If these signals are OK then the ALU V.C PAL is probably bad. If these signals are bad, check for a open etch, as these same signals were checked in the previous ALU N.Z test at that PAL. The rest of the data path after the ALU V.C PAL has been tested previously by the console.

Error 2 - Same as error 1 except C LONG should be high and V LONG should be high going into the ALU V.C PAL.

Error 3 - Same as error 1 except C LONG should be low and V LONG should be high going into the ALU V.C PAL.

Error 4 - Same as error 1 except C LONG should be low and V LONG should be low going into the ALU V.C PAL.

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 17 - ALU V.C B 3-MAR-82 L 5
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 17 - ALU V.C Byte, Word, Long Tests (M8390 CPU Module)

Fiche 2 Frame L5
Sequence 269
Rev 01.00
Page 263

:12242 : Error 5 - This error indicates a problem with the word data type. First
:12243 : check the C WORD H or V WORD H going into the ALU V.C PAL. C should
:12244 : be high and V should be low during the ADD LS TO WR instruction. If
:12245 : these are incorrect, then check the output from the 2901. Suspect a
:12246 : bad 2901 if the error is found there. If C and V into the pal are
:12247 : OK, then check the inputs LOAD ALU CC L for a high, ALU CC F0 H for a
:12248 : high, and ALU CC F1 H for a low. If these signals are OK then the
:12249 : ALU V.C PAL is probably bad. If these signals are bad, check for a
:12250 : open etch, as these same signals were checked in the previous ALU N.Z
:12251 : test at that PAL. The rest of the data path after the ALU V.C PAL has
:12252 : been tested previously by the console.

:12253 :
:12254 : Error 6 - Same as error 5 except C WORD should be high and V WORD
:12255 : should be high going into the ALU V.C PAL.

:12256 :
:12257 : Error 7 - Same as error 5 except C WORD should be low and V WORD
:12258 : should be high going into the ALU V.C PAL.

:12259 :
:12260 : Error 8 - Same as error 5 except C WORD should be low and V WORD
:12261 : should be low going into the ALU V.C PAL.

:12262 :
:12263 : Error 9 - This error indicates a problem with the byte data type. First
:12264 : check the C BYTE H or V BYTE H going into the ALU V.C PAL. C should
:12265 : be high and V should be low during the ADD LS TO WR instruction. If
:12266 : these are incorrect, then check the output from the 2901. Suspect a
:12267 : bad 2901 if the error is found there. If C and V into the pal are
:12268 : OK, then suspect the ALU V.C PAL itself.

:12269 :
:12270 : Error A - Same as error 9 except C BYTE should be high and V BYTE
:12271 : should be high going into the ALU V.C PAL.

:12272 :
:12273 : Error B - Same as error 9 except C BYTE should be low and V BYTE
:12274 : should be high going into the ALU V.C PAL.

:12275 :
:12276 : Error C - Same as error 9 except C BYTE should be low and V BYTE
:12277 : should be low going into the ALU V.C PAL.

:12278 :
:12279 :
:12280 : T.17:

U 31CB, B65E,15 :12281 MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
U 31CC, 3E80,15 :12282 MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
:12283 : BIT15 INDICATES START OF TEST TO CONSOLE
U 31CD, 10E0,15 :12284 MISC [SET.CP.ATTN] ;ISSUE CPU ATTN TO CONSOLE
:12285 :
WAIT.T17.0: :12286 JMP [WAIT.T17.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 31CE, 8B1C,E4 :12287 MOV LS[ONES] TO WR[0]
U 31CF, B69E,15 :12288 BIC LS[BIT0] TO WR[0]
U 31D0, 4540,15 :12289 BIC LS[BIT1] TO WR[0]
U 31D1, C542,15 :12290 MOV WR[0] TO LS[ERROR.MASK] ;ERROR MASK = #FFFFFFFC
U 31D2, 3E8A,15 :12291 CLR LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
U 31D3, 65A0,15 :12292 CLR WR[0]
U 31D4, 2F80,15 :12293 INC WR[0] ;SET WRO TO 1
U 31D5, 2040,15 :12294 MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
U 31D6, BE82,15 :12295 ROL WR[0] ;SET WRC TO 2
U 31D7, A3C0,15 :12296 MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
U 31D8, 3E8C,15 :12296

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

M 5
TEST 17 - ALU V.C B 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 17 - ALU V.C Byte, Word, Long Tests (M8390 CPU Module)

Fiche 2 Frame M5

Sequence 270

Rev 01.00 Page 264

```
U 31D9, 2040,15 :12297      INC WR[0]
U 31DA, BEFC,15 :12298      MOV WR[0] TO LS[SIZE]      ;SIZE REG = LONGWORD (11B)
                                LOOP.T17.1:
U 31DB, 369F,15 :12299      MOV LS[ONES] TO WR[2]      ;WR2 = #FFFFFFFF
                                ADD LS[BIT27] TO WR[2],
U 31DC, 4077,55 :12300      DT(SIZE)&SET.ALU.CC
U 31DD, 3640,95 :12301      MOV LS[BIT0] TO WR[1]      ;EXPECTED DATA = BIT0 SET
U 31DE, B7FC,15 :12302      MOV LS[ALU.CC] TO WR[0]      ;GET CC'S
U 31DF, 0869,3C :12303      JSR [CHECK.RESULT]      ;CHECK FOR C SET, V CLEAR
U 31E0, 0B1D,B4 :12304      JMP [LOOP.T17.1]      ;LOOP ON ERROR IF ENABLED
U 31E1, 369F,15 :12305      MOV LS[ONES] TO WR[2]      ;WR2 = #FFFFFFFF
                                ADD LS[BIT28] TO WR[2],
U 31E2, C079,55 :12306      DT(SIZE)&SET.ALU.CC
U 31E3, 3640,95 :12307      MOV LS[BIT0] TO WR[1]      ;EXPECTED DATA = BIT0 SET
U 31E4, B7FC,15 :12308      MOV LS[ALU.CC] TO WR[0]      ;GET CC'S
U 31E5, 0869,3C :12309      JSR [CHECK.RESULT]      ;CHECK FOR C SET, V CLEAR
U 31E6, 0B1D,B4 :12310      JMP [LOOP.T17.1]      ;LOOP ON ERROR IF ENABLED
U 31E7, 369F,15 :12311      MOV LS[ONES] TO WR[2]      ;WR2 = #FFFFFFFF
                                ADD LS[BIT29] TO WR[2],
U 31E8, 407B,55 :12312      DT(SIZE)&SET.ALU.CC
U 31E9, 3640,95 :12313      MOV LS[BIT0] TO WR[1]      ;EXPECTED DATA = BIT0 SET
U 31EA, B7FC,15 :12314      MOV LS[ALU.CC] TO WR[0]      ;GET CC'S
U 31EB, 0869,3C :12315      JSR [CHECK.RESULT]      ;CHECK FOR C SET, V CLEAR
U 31EC, 0B1D,B4 :12316      JMP [LOOP.T17.1]      ;LOOP ON ERROR IF ENABLED
U 31ED, 369F,15 :12317      MOV LS[ONES] TO WR[2]      ;WR2 = #FFFFFFFF
                                ADD LS[BIT30] TO WR[2],
U 31EE, 407D,55 :12318      DT(SIZE)&SET.ALU.CC
U 31EF, 3640,95 :12319      MOV LS[BIT0] TO WR[1]      ;EXPECTED DATA = BIT0 SET
U 31F0, B7FC,15 :12320      MOV LS[ALU.CC] TO WR[0]      ;GET CC'S
U 31F1, 0869,3C :12321      JSR [CHECK.RESULT]      ;CHECK FOR C SET, V CLEAR
U 31F2, 0B1D,B4 :12322      JMP [LOOP.T17.1]      ;LOOP ON ERROR IF ENABLED
U 31F3, 8870,5C :12323      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 2
                                LOOP.T17.2:
U 31F4, 369F,15 :12324      MOV LS[ONES] TO WR[2]      ;WR2 = #FFFFFFFF
                                ADD LS[BIT31] TO WR[2],
U 31F5, C07F,55 :12325      DT(SIZE)&SET.ALU.CC
U 31F6, 3640,95 :12326      MOV LS[BIT0] TO WR[1]
U 31F7, C742,95 :12327      BIS LS[BIT1] TO WR[1]      ;EXPECTED DATA = BIT0 & BIT1 SET
U 31F8, B7FC,15 :12328      MOV LS[ALU.CC] TO WR[0]      ;GET CC'S
U 31F9, 0869,3C :12329      JSR [CHECK.RESULT]      ;CHECK FOR C SET, V SET
U 31FA, 8B1F,44 :12330      JMP [LOOP.T17.2]      ;LOOP ON ERROR IF ENABLED
U 31FB, 8870,5C :12331      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 3
                                LOOP.T17.3:
U 31FC, 369F,15 :12332      MOV LS[ONES] TO WR[2]
U 31FD, 457F,15 :12333      BIC LS[BIT31] TO WR[2]      ;WR2 = #7FFFFFFF
                                ADD LS[BIT27] TO WR[2],
U 31FE, 4077,55 :12334      DT(SIZE)&SET.ALU.CC
U 31FF, B642,95 :12335      MOV LS[BIT1] TO WR[1]      ;EXPECTED DATA = BIT1 SET
U 3200, B7FC,15 :12336      MOV LS[ALU.CC] TO WR[0]      ;GET CC'S
U 3201, 0869,3C :12337      JSR [CHECK.RESULT]      ;CHECK FOR C CLEAR, V SET
U 3202, 0B1F,C4 :12338      JMP [LOOP.T17.3]      ;LOOP ON ERROR IF ENABLED
U 3203, 369F,15 :12339      MOV LS[ONES] TO WR[2]
U 3204, 457F,15 :12340      BIC LS[BIT31] TO WR[2]      ;WR2 = #7FFFFFFF
                                ADD LS[BIT28] TO WR[2],
U 3205, C079,55 :12341      DT(SIZE)&SET.ALU.CC
```

U 3206, B642.95	:12352	MOV LS[BIT1] TO WR[1]	:EXPECTED DATA = BIT1 SET
U 3207, B7FC.15	:12353	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 3208, 0869.3C	:12354	JSR [CHECK.RESULT]	:CHECK FOR C CLEAR, V SET
U 3209, 0B1F.C4	:12355	JMP [LOOP.T17.3]	:LOOP ON ERROR IF ENABLED
U 320A, 369F.15	:12356	MOV LS[ONES] TO WR[2]	
U 320B, 457F.15	:12357	BIC LS[BIT31] TO WR[2]	:WR2 = #7FFFFFFF
	:12358	ADD LS[BIT29] TO WR[2],	
U 320C, 407B.55	:12359	DT(SIZE)&SET.ALU.CC	
U 320D, B642.95	:12360	MOV LS[BIT1] TO WR[1]	:EXPECTED DATA = BIT1 SET
U 320E, B7FC.15	:12361	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 320F, 0869.3C	:12362	JSR [CHECK.RESULT]	:CHECK FOR C CLEAR, V SET
U 3210, 0B1F.C4	:12363	JMP [LOOP.T17.3]	:LOOP ON ERROR IF ENABLED
U 3211, 369F.15	:12364	MOV LS[ONES] TO WR[2]	
U 3212, 457F.15	:12365	BIC LS[BIT31] TO WR[2]	:WR2 = #7FFFFFFF
	:12366	ADD LS[BIT30] TO WR[2],	
U 3213, 407D.55	:12367	DT(SIZE)&SET.ALU.CC	
U 3214, B642.95	:12368	MOV LS[BIT1] TO WR[1]	:EXPECTED DATA = BIT1 SET
U 3215, B7FC.15	:12369	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 3216, 0869.3C	:12370	JSR [CHECK.RESULT]	:CHECK FOR C CLEAR, V SET
U 3217, 0B1F.C4	:12371	JMP [LOOP.T17.3]	:LOOP ON ERROR IF ENABLED
U 3218, 8870.5C	:12372	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 4
	:12373	LOOP.T17.4:	
U 3219, 369F.15	:12374	MOV LS[ONES] TO WR[2]	
U 321A, 457F.15	:12375	BIC LS[BIT31] TO WR[2]	:WR2 = #7FFFFFFF
	:12376	ADD LS[BIT31] TO WR[2],	
U 321B, C07F.55	:12377	DT(SIZE)&SET.ALU.CC	
U 321C, 2F82.95	:12378	CLR WR[1]	:EXPECTED DATA = 0
U 321D, B7FC.15	:12379	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 321E, 0869.3C	:12380	JSR [CHECK.RESULT]	:CHECK FOR C CLEAR, V CLEAR
U 321F, 8B21.94	:12381	JMP [LOOP.T17.4]	:LOOP ON ERROR IF ENABLED
U 3220, B640.15	:12382	MOV LS[BIT0] TO WR[0]	
U 3221, BEFC.15	:12383	MOV WR[0] TO LS[SIZE]	:SIZE = WORD (01B)
U 3222, 8870.5C	:12384	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 5
	:12385	LOOP.T17.5:	
U 3223, B629.15	:12386	MOV LS[FFFF] TO WR[2]	:WR2 = #FFFF
	:12387	ADD LS[BIT11] TO WR[2],	
U 3224, C057.55	:12388	DT(SIZE)&SET.ALU.CC	
U 3225, 3640.95	:12389	MOV LS[BIT0] TO WR[1]	:EXPECTED DATA = BIT0 SET
U 3226, B7FC.15	:12390	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 3227, 0869.3C	:12391	JSR [CHECK.RESULT]	:CHECK FOR C SET, V CLEAR
U 3228, 8B22.34	:12392	JMP [LOOP.T17.5]	:LOOP ON ERROR IF ENABLED
U 3229, B629.15	:12393	MOV LS[FFFF] TO WR[2]	:WR2 = #FFFF
	:12394	ADD LS[BIT12] TO WR[2],	
U 322A, 4059.55	:12395	DT(SIZE)&SET.ALU.CC	
U 322B, 3640.95	:12396	MOV LS[BIT0] TO WR[1]	:EXPECTED DATA = BIT0 SET
U 322C, B7FC.15	:12397	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 322D, 0869.3C	:12398	JSR [CHECK.RESULT]	:CHECK FOR C SET, V CLEAR
U 322E, 8B22.34	:12399	JMP [LOOP.T17.5]	:LOOP ON ERROR IF ENABLED
U 322F, B629.15	:12400	MOV LS[FFFF] TO WR[2]	:WR2 = #FFFF
	:12401	ADD LS[BIT13] TO WR[2],	
U 3230, C05B.55	:12402	DT(SIZE)&SET.ALU.CC	
U 3231, 3640.95	:12403	MOV LS[BIT0] TO WR[1]	:EXPECTED DATA = BIT0 SET
U 3232, B7FC.15	:12404	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 3233, 0869.3C	:12405	JSR [CHECK.RESULT]	:CHECK FOR C SET, V CLEAR
U 3234, 8B22.34	:12406	JMP [LOOP.T17.5]	:LOOP ON ERROR IF ENABLED

U 3235, B629,15	:12407	MOV LS[FFFF] TO WR[2]	;WR2 = #FFFF
	:12408	ADD LS[BIT14] TO WR[2],	
U 3236, C05D,55	:12409	DT(SIZE)&SET.ALU.CC	
U 3237, 3640,95	:12410	MOV LS[BIT0] TO WR[1]	;EXPECTED DATA = BIT0 SET
U 3238, B7FC,15	:12411	MOV LS[ALU.CC] TO WR[0]	;GET CC'S
U 3239, 0869,3C	:12412	JSR [CHECK.RESULT]	;CHECK FOR C SET, V CLEAR
U 323A, 8B22,34	:12413	JMP [LOOP.T17.5]	;LOOP ON ERROR IF ENABLED
U 323B, 8870,5C	:12414	JSR [INC.EN]	;INCREMENT ERROR NUMBER TO 6
	:12415	LOOP.T17.6:	
U 323C, B629,15	:12416	MOV LS[FFFF] TO WR[2]	;WR2 = #FFFF
	:12417	ADD LS[BIT15] TO WR[2],	
U 323D, 405F,55	:12418	DT(SIZE)&SET.ALU.CC	
U 323E, 3640,95	:12419	MOV LS[BIT0] TO WR[1]	
U 323F, C742,95	:12420	BIS LS[BIT1] TO WR[1]	;EXPECTED DATA = BIT0 & BIT1 SET
U 3240, B7FC,15	:12421	MOV LS[ALU.CC] TO WR[0]	;GET CC'S
U 3241, 0869,3C	:12422	JSR [CHECK.RESULT]	;CHECK FOR C SET, V SET
U 3242, 0B23,C4	:12423	JMP [LOOP.T17.6]	;LOOP ON ERROR IF ENABLED
U 3243, 8870,5C	:12424	JSR [INC.EN]	;INCREMENT ERROR NUMBER TO 7
	:12425	LOOP.T17.7:	
U 3244, 369F,15	:12426	MOV LS[ONES] TO WR[2]	
U 3245, C55F,15	:12427	BIC LS[BIT15] TO WR[2]	;WR2 = #7FFF
	:12428	ADD LS[BIT11] TO WR[2],	
U 3246, C057,55	:12429	DT(SIZE)&SET.ALU.CC	
U 3247, B642,95	:12430	MOV LS[BIT1] TO WR[1]	;EXPECTED DATA = BIT1 SET
U 3248, B7FC,15	:12431	MOV LS[ALU.CC] TO WR[0]	;GET CC'S
U 3249, 0869,3C	:12432	JSR [CHECK.RESULT]	;CHECK FOR C CLEAR, V SET
U 324A, 0B24,44	:12433	JMP [LOOP.T17.7]	;LOOP ON ERROR IF ENABLED
U 324B, B629,15	:12434	MOV LS[FFFF] TO WR[2]	
U 324C, C55F,15	:12435	BIC LS[BIT15] TO WR[2]	;WR2 = #7FFF
	:12436	ADD LS[BIT12] TO WR[2],	
U 324D, 4059,55	:12437	DT(SIZE)&SET.ALU.CC	
U 324E, B642,95	:12438	MOV LS[BIT1] TO WR[1]	;EXPECTED DATA = BIT1 SET
U 324F, B7FC,15	:12439	MOV LS[ALU.CC] TO WR[0]	;GET CC'S
U 3250, 0869,3C	:12440	JSR [CHECK.RESULT]	;CHECK FOR C CLEAR, V SET
U 3251, 0B24,44	:12441	JMP [LOOP.T17.7]	;LOOP ON ERROR IF ENABLED
U 3252, B629,15	:12442	MOV LS[FFFF] TO WR[2]	
U 3253, C55F,15	:12443	BIC LS[BIT15] TO WR[2]	;WR2 = #7FFF
	:12444	ADD LS[BIT13] TO WR[2],	
U 3254, C05B,55	:12445	DT(SIZE)&SET.ALU.CC	
U 3255, B642,95	:12446	MOV LS[BIT1] TO WR[1]	;EXPECTED DATA = BIT1 SET
U 3256, B7FC,15	:12447	MOV LS[ALU.CC] TO WR[0]	;GET CC'S
U 3257, 0869,3C	:12448	JSR [CHECK.RESULT]	;CHECK FOR C CLEAR, V SET
U 3258, 0B24,44	:12449	JMP [LOOP.T17.7]	;LOOP ON ERROR IF ENABLED
U 3259, B629,15	:12450	MOV LS[FFFF] TO WR[2]	
U 325A, C55F,15	:12451	BIC LS[BIT15] TO WR[2]	;WR2 = #7FFF
	:12452	ADD LS[BIT14] TO WR[2],	
U 325B, C05D,55	:12453	DT(SIZE)&SET.ALU.CC	
U 325C, B642,95	:12454	MOV LS[BIT1] TO WR[1]	;EXPECTED DATA = BIT1 SET
U 325D, B7FC,15	:12455	MOV LS[ALU.CC] TO WR[0]	;GET CC'S
U 325E, 0869,3C	:12456	JSR [CHECK.RESULT]	;CHECK FOR C CLEAR, V SET
U 325F, 0B24,44	:12457	JMP [LOOP.T17.7]	;LOOP ON ERROR IF ENABLED
U 3260, 8870,5C	:12458	JSR [INC.EN]	;INCREMENT ERROR NUMBER TO 8
	:12459	LOOP.T17.8:	
U 3261, B629,15	:12460	MOV LS[FFFF] TO WR[2]	
U 3262, C55F,15	:12461	BIC LS[BIT15] TO WR[2]	;WR2 = #7FFF

U 3263, 405F,55	:12462	ADD LS[BIT15] TO WR[2],	
U 3264, 2F82,95	:12463	DT(SIZE)&SET.ALU.CC	
U 3265, B7FC,15	:12464	CLR WR[1]	:EXPECTED DATA = 0
U 3266, 0869,3C	:12465	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 3267, 8B26,14	:12466	JSR [CHECK.RESULT]	:CHECK FOR C CLEAR, V CLEAR
U 3268, 65FC,15	:12467	JMP [LOOP.T17.8]	:LOOP ON ERROR IF ENABLED
U 3269, 8870,5C	:12468	CLR LS[SIZE]	:SIZE = BYTE (00B)
	:12469	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 9
	:12470	LOOP.T17.9:	
U 326A, 3627,15	:12471	MOV LS[0FF] TO WR[2]	:WR2 = 0FF
	:12472	ADD LS[BIT3] TO WR[2],	
U 326B, 4047,55	:12473	DT(SIZE)&SET.ALU.CC	
U 326C, 3640,95	:12474	MOV LS[BIT0] TO WR[1]	:EXPECTED DATA = BIT0 SET
U 326D, B7FC,15	:12475	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 326E, 0869,3C	:12476	JSR [CHECK.RESULT]	:CHECK FOR C SET, V CLEAR
U 326F, 0B26,A4	:12477	JMP [LOOP.T17.9]	:LOOP ON ERROR IF ENABLED
U 3270, 3627,15	:12478	MOV LS[0FF] TO WR[2]	:WR2 = 0FF
	:12479	ADD LS[BIT4] TO WR[2],	
U 3271, C049,55	:12480	DT(SIZE)&SET.ALU.CC	
U 3272, 3640,95	:12481	MOV LS[BIT0] TO WR[1]	:EXPECTED DATA = BIT0 SET
U 3273, B7FC,15	:12482	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 3274, 0869,3C	:12483	JSR [CHECK.RESULT]	:CHECK FOR C SET, V CLEAR
U 3275, 0B26,A4	:12484	JMP [LOOP.T17.9]	:LOOP ON ERROR IF ENABLED
U 3276, 3627,15	:12485	MOV LS[0FF] TO WR[2]	:WR2 = 0FF
	:12486	ADD LS[BIT5] TO WR[2],	
U 3277, 404B,55	:12487	DT(SIZE)&SET.ALU.CC	
U 3278, 3640,95	:12488	MOV LS[BIT0] TO WR[1]	:EXPECTED DATA = BIT0 SET
U 3279, B7FC,15	:12489	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 327A, 0869,3C	:12490	JSR [CHECK.RESULT]	:CHECK FOR C SET, V CLEAR
U 327B, 0B26,A4	:12491	JMP [LOOP.T17.9]	:LOOP ON ERROR IF ENABLED
U 327C, 3627,15	:12492	MOV LS[0FF] TO WR[2]	:WR2 = 0FF
	:12493	ADD LS[BIT6] TO WR[2],	
U 327D, 404D,55	:12494	DT(SIZE)&SET.ALU.CC	
U 327E, 3640,95	:12495	MOV LS[BIT0] TO WR[1]	:EXPECTED DATA = BIT0 SET
U 327F, B7FC,15	:12496	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 3280, 0869,3C	:12497	JSR [CHECK.RESULT]	:CHECK FOR C SET, V CLEAR
U 3281, 0B26,A4	:12498	JMP [LOOP.T17.9]	:LOOP ON ERROR IF ENABLED
U 3282, 8870,5C	:12499	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO A
	:12500	LOOP.T17.A:	
U 3283, 3627,15	:12501	MOV LS[0FF] TO WR[2]	:WR2 = 0FF
	:12502	ADD LS[BIT7] TO WR[2],	
U 3284, C04F,55	:12503	DT(SIZE)&SET.ALU.CC	
U 3285, 3640,95	:12504	MOV LS[BIT0] TO WR[1]	
U 3286, C742,95	:12505	BIS LS[BIT1] TO WR[1]	:EXPECTED DATA = BIT0 & BIT1 SET
U 3287, B7FC,15	:12506	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 3288, 0869,3C	:12507	JSR [CHECK.RESULT]	:CHECK FOR C SET, V SET
U 3289, 8B28,34	:12508	JMP [LOOP.T17.A]	:LOOP ON ERROR IF ENABLED
U 328A, 8870,5C	:12509	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO B
	:12510	LOOP.T17.B:	
U 328B, 369F,15	:12511	MOV LS[ONES] TO WR[2]	
U 328C, 454F,15	:12512	BIC LS[BIT7] TO WR[2]	:WR2 = 07F
	:12513	ADD LS[BIT3] TO WR[2],	
U 328D, 4047,55	:12514	DT(SIZE)&SET.ALU.CC	
U 328E, B642,95	:12515	MOV LS[BIT1] TO WR[1]	:EXPECTED DATA = BIT1 SET
U 328F, B7FC,15	:12516	MOV LS[ALU.CC] TO WR[0]	:GET CC'S

U 3290, 0869,3C	:12517	JSR [CHECK.RESULT]	:CHECK FOR C CLEAR, V SET
U 3291, 0B28,B4	:12518	JMP [LOOP.T17.B]	:LOOP ON ERROR IF ENABLED
U 3292, 3627,15	:12519	MOV LS[7FF] TO WR[2]	
U 3293, 454F,15	:12520	BIC LS[BIT7] TO WR[2]	:WR2 = #7F
	:12521	ADD LS[BIT4] TO WR[2],	
U 3294, C049,55	:12522	DT(SIZE)&SET.ALU.CC	
U 3295, B642,95	:12523	MOV LS[BIT1] TO WR[1]	:EXPECTED DATA = BIT1 SET
U 3296, B7FC,15	:12524	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 3297, 0869,3C	:12525	JSR [CHECK.RESULT]	:CHECK FOR C CLEAR, V SET
U 3298, 0B28,B4	:12526	JMP [LOOP.T17.B]	:LOOP ON ERROR IF ENABLED
U 3299, 3627,15	:12527	MOV LS[7FF] TO WR[2]	
U 329A, 454F,15	:12528	BIC LS[BIT7] TO WR[2]	:WR2 = #7F
	:12529	ADD LS[BIT5] TO WR[2],	
U 329B, 404B,55	:12530	DT(SIZE)&SET.ALU.CC	
U 329C, B642,95	:12531	MOV LS[BIT1] TO WR[1]	:EXPECTED DATA = BIT1 SET
U 329D, B7FC,15	:12532	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 329E, 0869,3C	:12533	JSR [CHECK.RESULT]	:CHECK FOR C CLEAR, V SET
U 329F, 0B28,B4	:12534	JMP [LOOP.T17.B]	:LOOP ON ERROR IF ENABLED
U 32A0, 3627,15	:12535	MOV LS[7FF] TO WR[2]	
U 32A1, 454F,15	:12536	BIC LS[BIT7] TO WR[2]	:WR2 = #7F
	:12537	ADD LS[BIT6] TO WR[2],	
U 32A2, 404D,55	:12538	DT(SIZE)&SET.ALU.CC	
U 32A3, B642,95	:12539	MOV LS[BIT1] TO WR[1]	:EXPECTED DATA = BIT1 SET
U 32A4, B7FC,15	:12540	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 32A5, 0869,3C	:12541	JSR [CHECK.RESULT]	:CHECK FOR C CLEAR, V SET
U 32A6, 0B28,B4	:12542	JMP [LOOP.T17.B]	:LOOP ON ERROR IF ENABLED
U 32A7, 8870,5C	:12543	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO C
	:12544	LOOP.T17.C:	
U 32A8, 3627,15	:12545	MOV LS[7FF] TO WR[2]	
U 32A9, 454F,15	:12546	BIC LS[BIT7] TO WR[2]	:WR2 = #7F
	:12547	ADD LS[BIT7] TO WR[2],	
U 32AA, C04F,55	:12548	DT(SIZE)&SET.ALU.CC	
U 32AB, 2F82,95	:12549	CLR WR[1]	:EXPECTED DATA = 0
U 32AC, B7FC,15	:12550	MOV LS[ALU.CC] TO WR[0]	:GET CC'S
U 32AD, 0869,3C	:12551	JSR [CHECK.RESULT]	:CHECK FOR C CLEAR, V CLEAR
U 32AE, 8B2A,84	:12552	JMP [LOOP.T17.C]	:LOOP ON ERROR IF ENABLED
	:12553	END.T17:	

Page "TEST 18 - ALU C L, HALF CARRY L, and NOP Tests (M8390 CPU Module)"

++

Test Description:

This test checks the HALF CARRY L and ALU C L condition codes for all three data types: byte, word, and longword. The size reg will be loaded and used to control the data type for all three types. The data path is as follows: WR 2 is loaded from LS with a data pattern and an ADD LS TO WR is executed with SET ALU CC's enabled. Then the ALU C L and HALF CARRY L bits are enabled onto the D bus and written into WR 0. WR 1 is written with expected data from LS and the 2 condition code bits are checked. These bits come off the D BUS in the following manner: ALU C L comes off D D05 and D06, HALF CARRY L comes off D D01 and D D02, and the other bits from D D00 through D D07 are forced low. All 8 bits will be checked to assure that the unused bits are low. After this check, the NOP function of the condition codes is checked. This involves loading the condition codes as before, but then executing another instruction that would change the codes, but with the CC field of the u-word equal to 0. This should prevent loading the new 2901 status bits. ALU N, ALU Z, ALU V, ALU C H, and ALU C L are checked for no change. HALF CARRY L is the exception in that it should follow CARRY 4 H from the 2901 regardless of the CC field. The data patterns used are all ones and shifting ones, all ones and all zeros with carry in on and off, and other combinations necessary to fully test the ALU C L and HALF CARRY bits. These patterns will also assure that the data type selected is enabling the correct 2901 status bits.

Assumptions:

It is assumed that the previous 2901 tests and the previous condition code tests have been run successfully.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load SIZE REG with 11(B) to indicate longword data type.
- 3) Load WR 2 with all ones and then execute ADD LS TO WR with data of 0 from LS. Also enable loading of ALU CC's in this instruction with CC field of u-word=2.
- 4) Load WR 1 with expected data, load WR 0 with ALU C L and HALF CARRY L (LS=7D), and check for ALU C L set, HALF CARRY L set, and the other 4 bits clear.
- 5) Repeat step 3 using data of a 1 in bit 31 from LS. Load WR 1 with expected data, load WR 0 with ALU C L and HALF CARRY L, and check for HALF CARRY L set and all other bits clear.
- 6) Load SIZE REG with 01(B) to indicate word data type.
- 7) Repeat steps 3 through 5 above with data of FFFF for step 3. Substitute bit 15 for bit 31 in step 5.
- 8) Load SIZE REG with 00(B) to indicate byte data type.
- 9) Repeat steps 3 through 5 above with data of FF for step 3. Substitute bit 7 for bit 31 in step 5.
- 10) Load WR 2 with F(H) and then execute ADD LS TO WR + 1 with data of


```

:12609 : 0's from LS. Also enable loading of ALU CC's in this instruction
:12610 : with CC field of u-word=2.
:12611 : 11) Load WR 1 with expected data, load WR 0 with ALU C L and HALF
:12612 : CARRY L, and check for ALU C L set, HALF CARRY L clear, and all
:12613 : other bits clear.
:12614 : 12) Load WR 2 with F(H) and then execute ADD LS TO WR with data of
:12615 : shifting 1's from LS starting at bit 0. Also enable loading of
:12616 : ALU CC's in this instruction with CC field of u-word=2.
:12617 : 13) Load WR 1 with expected data, load WR 0 with ALU C L and HALF
:12618 : CARRY L, and check for ALU C L set, HALF CARRY L clear, and all
:12619 : other bits clear.
:12620 : 14) Repeat steps 12 and 13, shifting the 1 from LS left until an add
:12621 : with data of a 1 in bit 3 from LS has been tested.
:12622 : 15) Load WR 2 with all 1's, execute ADD LS TO WR with a 1 in bit 0 as
:12623 : data from LS, with CC field of u-word=1 (load ALU CC's, longword).
:12624 : This will set ALU C H, ALU Z, and clear HALF CARRY L, ALU N, ALU
:12625 : V, and ALU C L.
:12626 : 16) Load WR 1 with 7FFFFFFF, and execute ADD LS TO WR with a 1 in bit
:12627 : 4 as data from LS. CC field of u-word=0 (do not load ALU CC's).
:12628 : This has the effect of toggling all bits, but the no load mode
:12629 : should prevent this, except for HALF CARRY L.
:12630 : 17) Load WR 0 with ALU C L and HALF CARRY L, load WR 1 with expected
:12631 : data, and check for HALF CARRY L set, and ALU C L and other bits
:12632 : clear.
:12633 : 18) Change error mask to check bits 0-3 only, load WR 0 with ALU C H,
:12634 : ALU N, ALU Z, and ALU V, and load WR 1 with expected data. Check
:12635 : for ALU C H and ALU Z set, others clear.
:12636 : 19) Load WR 2 with 7FFFFFFF, execute ADD LS TO WR with a 1 in bit 4 as
:12637 : data from LS, with CC field of u-word=1 (load ALU CC's, longword).
:12638 : This will clear ALU C H and ALU Z, and set the other condition
:12639 : code bits.
:12640 : 20) Load WR 1 with all 1's, and execute ADD LS TO WR with a 1 in bit
:12641 : 0 as data from LS. CC field of u-word=0 (do not load ALU CC's).
:12642 : This has the effect of toggling all bits, but the no load mode
:12643 : should prevent this, except for HALF CARRY L.
:12644 : 21) Load WR 0 with ALU C H, ALU N, ALU Z, and ALU V, load WR 1 with
:12645 : expected data, and check for ALU C H and ALU Z clear, and the
:12646 : other 2 bits set.
:12647 : 22) Change error mask to check bits 0-7, load WR 0 with ALU C L and
:12648 : HALF CARRY L, and load WR 1 with expected data. Check for ALU C
:12649 : L set, and HALF CARRY L and other bits clear.
:12650 :
:12651 :
:12652 :
:12653 :
:12654 :
:12655 :
:12656 :
:12657 :
:12658 :
:12659 :
:12660 :
:12661 :
:12662 :
:12663 :

```

Errors:

- Error 1 - ALU C L and/or HALF CARRY L failed. ALU C L and HALF CARRY L should have set. All other bits (0, 3, 4 and 7) should have been clear. Data type=longword.
- Error 2 - ALU C L and/or HALF CARRY L failed. ALU C L should have cleared, HALF CARRY L should have set. All other bits (0, 3, 4 and 7) should have been clear. Data type=longword.
- Error 3 - ALU C L and/or HALF CARRY L failed. ALU C L should have set, HALF CARRY L should have set. All other bits (0, 3, 4 and 7) should have been clear. Data type=word.
- Error 4 - ALU C L and/or HALF CARRY L failed. ALU C L should have cleared, HALF CARRY L should have set. All other bits (0,

:12664 : 3, 4 and 7) should have been clear. Data type=word.
:12665 : Error 5 - ALU C L and/or HALF CARRY L failed. ALU C L should have set,
:12666 : HALF CARRY L should have set. All other bits (0, 3, 4 and 7)
:12667 : should have been clear. Data type=byte.
:12668 : Error 6 - ALU C L and/or HALF CARRY L failed. ALU C L should have
:12669 : cleared, HALF CARRY L should have set. All other bits (0,
:12670 : 3, 4 and 7) should have been clear. Data type=byte.
:12671 : Error 7 - ALU C L and/or HALF CARRY L failed. ALU C L should have set,
:12672 : HALF CARRY L should have cleared. All other bits (0, 3, 4
:12673 : and 7) should have been clear.
:12674 : Error 8 - Same as error 7.
:12675 : Error 9 - CC field=0 (no load ALU) failed to prevent ALU codes from
:12676 : changing. HALF CARRY L should have been set, ALU C L and
:12677 : other bits should have been low.
:12678 : Error A - CC field=0 (no load ALU) failed to prevent ALU codes from
:12679 : changing. ALU C H, and ALU Z should have been set, ALU N
:12680 : and ALU V should have been clear.
:12681 : Error B - CC field=0 (no load ALU) failed to prevent ALU codes from
:12682 : changing. ALU C H, and ALU Z should have been clear, ALU N
:12683 : and ALU V should have been set.
:12684 : Error C - CC field=0 (no load ALU) failed to prevent ALU codes from
:12685 : changing. ALU C L should have been high, HALF CARRY L and
:12686 : other bits should have been low.
:12687 :
:12688 :
:12689 :
:12690 :
:12691 :
:12692 :
:12693 :
:12694 :
:12695 :
:12696 :
:12697 :
:12698 :
:12699 :
:12700 :
:12701 :
:12702 :
:12703 :
:12704 :
:12705 :
:12706 :
:12707 :
:12708 :
:12709 :
:12710 :
:12711 :
:12712 :
:12713 :
:12714 :
:12715 :
:12716 :
:12717 :
:12718 :

Debug:

Error 1 - This error indicates a problem with a 2901, the ALU V.C PAL, or the data path out of that PAL. First check CARRY 32 H and CARRY 4 H for a low into the ALU V.C PAL during the ADD u-word. If these are incorrect, check their output at the 2901. The 2901 is suspect if the signal there is incorrect. If these inputs are OK, check ALU C L on pin 14 and HALF CARRY L on pin 15 for a high. The PAL is bad if these signals are incorrect. The data path to the D BUS for these signals is suspect if no problem is found on the output. During the u-word that outputs the result to WR 0, check the SEL lines on the 2X4 muxes for a 2 (pin 2 high, pin 14 low). On the 2X4 mux check the SEL for a 0 (low on pin 1). The enables should also be checked for a low. A problem here indicates a faulty REG CONTROL PAL. If the enable and selects are OK, check the ALU C L, HALF CARRY L, and grounds going into the MUXes. The MUX itself is bad if these inputs are OK.

Error 2 - Same as error 1 except CARRY 32 H should be high and ALU C L out of the ALU V.C PAL should be low.

Error 3 - If this is the first error, the ALU V.C PAL is the most likely suspect. The data path is the same as error 1 except CARRY 16 replaces CARRY 32.

Error 4 - If this is the first error, the ALU V.C PAL is the most likely suspect. The data path is the same as error 2 except CARRY 16 replaces CARRY 32.

Error 5 - Same as error 3 except CARRY 8 replaces CARRY 32.

Error 6 - Same as error 4 except CARRY 8 replaces CARRY 32.

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) TEST 18 - ALU C L, HALF CARRY L, and NOP Tests (M8390 CPU Module)
3-MAR-82 09:34:33

H 6

Fiche 2 Frame H6

Sequence 278
Rev 01.00

Page 272

:12719
:12720
:12721
:12722
:12723
:12724
:12725
:12726
:12727
:12728
:12729
:12730
:12731
:12732
:12733
:12734
:12735
:12736
:12737
:12738
:12739
:12740
:12741
:12742
:12743
:12744
:12745
:12746
U 32AF, B65E,15 :12747
U 32B0, 3E80,15 :12748
:12749
U 32B1, 10E0,15 :12750
:12751
U 32B2, 0B2B,24 :12752
U 32B3, DF26,15 :12753
U 32B4, 3E8A,15 :12754
U 32B5, 65A0,15 :12755
U 32B6, 2F80,15 :12756
U 32B7, 2040,15 :12757
U 32B8, BE82,15 :12758
U 32B9, A3C0,15 :12759
U 32BA, 3E8C,15 :12760
U 32BB, 2040,15 :12761
U 32BC, BEFC,15 :12762
:12763
U 32BD, 369F,15 :12764
U 32BE, B642,95 :12765
U 32BF, C744,95 :12766
U 32C0, 474A,95 :12767
U 32C1, 474C,95 :12768
:12769
U 32C2, C09D,55 :12770
U 32C3, 36FA,15 :12771
U 32C4, 0B69,3C :12772
U 32C5, 0B2B,D4 :12773

Error 7 - This is the first test of HALF CARRY L in the low state. Follow data path described in error 1, except CARRY 4 should be high and HALF CARRY L out of the ALU V.C PAL should be low.

Error 8 - Same as error 7, but the most likely suspect is the low 2901 if this is the first error.

Error 9 - This is the first test of the no load CC code (CC field of u-word=0). Check the inputs to the ALU V.C PAL for a high on both ALU CC F0 H and ALU CC F1 H during the ADD u-word with CC field=0. If these inputs are OK, then suspect the ALU V.C PAL. If a problem is found at the inputs, check the inputs to the CC CONTROL PAL for a low on CSR 05 H and CSR 06 H. If these are correct, but the outputs on pins 16 and 17 are not both high, the CC CONTROL PAL is faulty.

Error A - Same as error 9 except the ALU N.Z PAL is used and this PAL is the most likely suspect.

Error B - The ALU N.Z PAL is the most likely suspect. Data path is the same as error 9 except substitute the ALU N.Z PAL for the ALU V.C PAL.

Error C - The ALU V.C PAL is the most likely suspect. Data path is the same as error 9.

T.18:

WAIT.T18.0:

LOOP.T18.1:

MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN] ;BIT15 INDICATES START OF TEST TO CONSOLE
;ISSUE CPU ATTN TO CONSOLE
JMP [WAIT.T18.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
MCOM LS[FFF] TO WR[0]
MOV WR[0] TO LS[ERROR.MASK] ;ERROR MASK = #FFFFFFF0
CLR LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
CLR WR[0]
INC WR[0] ;SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
ROL WR[0] ;SET WRO TO 2
MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
INC WR[0]
MOV WR[0] TO LS[SIZE] ;SIZE = LONGWORD (11B)
MOV LS[ONES] TO WR[2] ;WR2 = #FFFFFFF
MOV LS[#2] TO WR[1]
BIS LS[#4] TO WR[1]
BIS LS[#20] TO WR[1]
BIS LS[#40] TO WR[1] ;WR1 = #66
ADD LS[ZERO] TO WR[2].
DT(SIZE)&SET.ALU.CC
MOV LS[DEC.CON] TO WR[0] ;GET D D00 TO D D07
JSR [CHECK.RESULT] ;CHECK FOR ALU C L SET, HALF CARRY L SET
JMP [LOOP.T18.1] ;LOOP ON ERROR IF ENABLED

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) TEST 18 - ALU C L, HALF CARRY L, and NOP Tests (M8390 CPU Module)
3-MAR-82 09:34:33
Fiche 2 Frame 16
ENKCA Micro-diagnostic for DAP module

Sequence 279
Rev 01.00
Page 273

```
U 32C6, 8870,5C :12774 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
:12775 LOOP.T18.2: ;WR2 = #FFFFFFF
U 32C7, 369F,15 :12776 MOV LS[ONES] TO WR[2]
U 32C8, B642,95 :12777 MOV LS[#2] TO WR[1]
U 32C9, C744,95 :12778 BIS LS[#4] TO WR[1] ;WR1 = #06
:12779 ADD LS[BIT31] TO WR[2],
U 32CA, C07F,55 :12780 DT(SIZE)&SET.ALU.CC
U 32CB, 36FA,15 :12781 MOV LS[DEC.CON] TO WR[0] ;GET D D00 TO D D07
U 32CC, 0869,3C :12782 JSR [CHECK.RESULT] ;CHECK FOR ALU C L CLEAR, HALF CARRY L SET
U 32CD, 8B2C,74 :12783 JMP [LOOP.T18.2] ;LOOP ON ERROR IF ENABLED
U 32CE, 8870,5C :12784 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 3
U 32CF, B640,15 :12785 MOV LS[#1] TO WR[0]
U 32D0, BEFC,15 :12786 MOV WR[0] TO LS[SIZE] ;SIZE = WORD (01B)
:12787 LOOP.T18.3: ;WR2 = #FFFF
U 32D1, B629,15 :12788 MOV LS[#FFFF] TO WR[2]
U 32D2, B642,95 :12789 MOV LS[#2] TO WR[1]
U 32D3, C744,95 :12790 BIS LS[#4] TO WR[1]
U 32D4, 474A,95 :12791 BIS LS[#20] TO WR[1]
U 32D5, 474C,95 :12792 BIS LS[#40] TO WR[1] ;WR1 = #66
:12793 ADD LS[ZERO] TO WR[2],
U 32D6, C09D,55 :12794 DT(SIZE)&SET.ALU.CC
U 32D7, 36FA,15 :12795 MOV LS[DEC.CON] TO WR[0] ;GET D D00 TO D D07
U 32D8, 0869,3C :12796 JSR [CHECK.RESULT] ;CHECK FOR ALU C L SET, HALF CARRY L SET
U 32D9, 0B2D,14 :12797 JMP [LOOP.T18.3] ;LOOP ON ERROR IF ENABLED
U 32DA, 8870,5C :12798 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 4
:12799 LOOP.T18.4: ;WR2 = #FFFF
U 32DB, B629,15 :12800 MOV LS[#FFFF] TO WR[2]
U 32DC, B642,95 :12801 MOV LS[#2] TO WR[1]
U 32DD, C744,95 :12802 BIS LS[#4] TO WR[1] ;WR1 = #06
:12803 ADD LS[BIT15] TO WR[2],
U 32DE, 405F,55 :12804 DT(SIZE)&SET.ALU.CC
U 32DF, 36FA,15 :12805 MOV LS[DEC.CON] TO WR[0] ;GET D D00 TO D D07
U 32E0, 0869,3C :12806 JSR [CHECK.RESULT] ;CHECK FOR ALU C L CLEAR, HALF CARRY L SET
U 32E1, 0B2D,B4 :12807 JMP [LOOP.T18.4] ;LOOP ON ERROR IF ENABLED
U 32E2, 8870,5C :12808 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 5
U 32E3, 65FC,15 :12809 CLR LS[SIZE] ;SIZE = BYTE (00B)
:12810 LOOP.T18.5: ;WR2 = #FF
U 32E4, 3627,15 :12811 MOV LS[#FF] TO WR[2]
U 32E5, B642,95 :12812 MOV LS[#2] TO WR[1]
U 32E6, C744,95 :12813 BIS LS[#4] TO WR[1]
U 32E7, 474A,95 :12814 BIS LS[#20] TO WR[1]
U 32E8, 474C,95 :12815 BIS LS[#40] TO WR[1] ;WR1 = #66
:12816 ADD LS[ZERO] TO WR[2],
U 32E9, C09D,55 :12817 DT(SIZE)&SET.ALU.CC
U 32EA, 36FA,15 :12818 MOV LS[DEC.CON] TO WR[0] ;GET D D00 TO D D07
U 32EB, 0869,3C :12819 JSR [CHECK.RESULT] ;CHECK FOR ALU C L SET, HALF CARRY L SET
U 32EC, 0B2E,44 :12820 JMP [LOOP.T18.5] ;LOOP ON ERROR IF ENABLED
U 32ED, 8870,5C :12821 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 6
:12822 LOOP.T18.6: ;WR2 = #FF
U 32EE, 3627,15 :12823 MOV LS[#FF] TO WR[2]
U 32EF, B642,95 :12824 MOV LS[#2] TO WR[1]
U 32F0, C744,95 :12825 BIS LS[#4] TO WR[1] ;WR1 = #06
:12826 ADD LS[BIT7] TO WR[2],
U 32F1, C04F,55 :12827 DT(SIZE)&SET.ALU.CC
U 32F2, 36FA,15 :12828 MOV LS[DEC.CON] TO WR[0] ;GET D D00 TO D D07
```



```

U 32F3, 0869,3C :12829      JSR [CHECK.RESULT]      ;CHECK FOR ALU C L CLEAR, HALF CARRY L SET
U 32F4, 0B2E,F4 :12830      JMP [LOOP.T18.6]      ;LOOP ON ERROR IF ENABLED
U 32F5, 8870,5C :12831      JSR [INC.EN]        ;INCREMENT ERROR NUMBER TO 7
                                LOOP.T18.7:
U 32F6, B649,15 :12832      MOV LS[#10] TO WR[2]
U 32F7, 4141,15 :12833      SUB LS[#1] FROM WR[2]      ;WR2 = #10 - #1 = #F
U 32F8, 364A,95 :12834      MOV LS[#20] TO WR[1]
U 32F9, 474C,95 :12835      BIS LS[#40] TO WR[1]      ;WR1 = #60
                                ADD (LS[ZERO] TO WR[2])+1,
U 32FA, C69D,55 :12836      DT(SIZE)&SET.ALU.CC
U 32FB, 36FA,15 :12837      MOV LS[DEC.CON] TO WR[0]      ;GET D D00 TO D D07
U 32FC, 0869,3C :12838      JSR [CHECK.RESULT]      ;CHECK FOR ALU C L SET, HALF CARRY L CLEAR
U 32FD, 0B2F,64 :12839      JMP [LOOP.T18.7]      ;LOOP ON ERROR IF ENABLED
U 32FE, 8870,5C :12840      JSR [INC.EN]        ;INCREMENT ERROR NUMBER TO 8
                                LOOP.T18.8:
U 32FF, B649,15 :12841      MOV LS[#10] TO WR[2]
U 3300, 4141,15 :12842      SUB LS[#1] FROM WR[2]      ;WR2 = #10 - #1 = #F
U 3301, 364A,95 :12843      MOV LS[#20] TO WR[1]
U 3302, 474C,95 :12844      BIS LS[#40] TO WR[1]      ;WR1 = #60
                                ADD LS[#1] TO WR[2],
U 3303, 4041,55 :12845      DT(SIZE)&SET.ALU.CC
U 3304, 36FA,15 :12846      MOV LS[DEC.CON] TO WR[0]      ;GET D D00 TO D D07
U 3305, 0869,3C :12847      JSR [CHECK.RESULT]      ;CHECK FOR ALU C L SET, HALF CARRY L CLEAR
U 3306, 0B2F,F4 :12848      JMP [LOOP.T18.8]      ;LOOP ON ERROR IF ENABLED
U 3307, B649,15 :12849      MOV LS[#10] TO WR[2]
U 3308, 4141,15 :12850      SUB LS[#1] FROM WR[2]      ;WR2 = #10 - #1 = #F
U 3309, 364A,95 :12851      MOV LS[#20] TO WR[1]
U 330A, 474C,95 :12852      BIS LS[#40] TO WR[1]      ;WR1 = #60
                                ADD LS[#2] TO WR[2],
U 330B, C043,55 :12853      DT(SIZE)&SET.ALU.CC
U 330C, 36FA,15 :12854      MOV LS[DEC.CON] TO WR[0]      ;GET D D00 TO D D07
U 330D, 0869,3C :12855      JSR [CHECK.RESULT]      ;CHECK FOR ALU C L SET, HALF CARRY L CLEAR
U 330E, 0B2F,F4 :12856      JMP [LOOP.T18.8]      ;LOOP ON ERROR IF ENABLED
U 330F, B649,15 :12857      MOV LS[#10] TO WR[2]
U 3310, 4141,15 :12858      SUB LS[#1] FROM WR[2]      ;WR2 = #10 - #1 = #F
U 3311, 364A,95 :12859      MOV LS[#20] TO WR[1]
U 3312, 474C,95 :12860      BIS LS[#40] TO WR[1]      ;WR1 = #60
                                ADD LS[#4] TO WR[2],
U 3313, C045,55 :12861      DT(SIZE)&SET.ALU.CC
U 3314, 36FA,15 :12862      MOV LS[DEC.CON] TO WR[0]      ;GET D D00 TO D D07
U 3315, 0869,3C :12863      JSR [CHECK.RESULT]      ;CHECK FOR ALU C L SET, HALF CARRY L CLEAR
U 3316, 0B2F,F4 :12864      JMP [LOOP.T18.8]      ;LOOP ON ERROR IF ENABLED
U 3317, B649,15 :12865      MOV LS[#10] TO WR[2]
U 3318, 4141,15 :12866      SUB LS[#1] FROM WR[2]      ;WR2 = #10 - #1 = #F
U 3319, 364A,95 :12867      MOV LS[#20] TO WR[1]
U 331A, 474C,95 :12868      BIS LS[#40] TO WR[1]      ;WR1 = #60
                                ADD LS[#8] TO WR[2],
U 331B, 4047,55 :12869      DT(SIZE)&SET.ALU.CC
U 331C, 36FA,15 :12870      MOV LS[DEC.CON] TO WR[0]      ;GET D D00 TO D D07
U 331D, 0869,3C :12871      JSR [CHECK.RESULT]      ;CHECK FOR ALU C L SET, HALF CARRY L CLEAR
U 331E, 0B2F,F4 :12872      JMP [LOOP.T18.8]      ;LOOP ON ERROR IF ENABLED
                                LOOP.T18.9:
U 331F, DF26,15 :12873      MCOM LS[FF] TO WR[0]
U 3320, 3E8A,15 :12874      MOV WR[0] TO LS[ERROR.MASK] ;ERROR MASK = #FFFFFF00
U 3321, 3647,15 :12875      MOV LS[#8] TO WR[2]
    
```

```

U 3322, 2045.15 :12884      INC WR[2]
U 3323, 3E83.15 :12885      MOV WR[2] TO LS[ERROR.NUMBER] ;ERROR NUMBER = 9
U 3324, 369F.15 :12886      MOV LS[ONES] TO WR[2] ;WR2 = #FFFFFFF
                                ADD LS[#1] TO WR[2]
U 3325, 4041.35 :12887      DT(LONG)&SET.ALU.CC
U 3326, 369E.95 :12888      MOV LS[ONES] TO WR[1]
U 3327, 457E.95 :12889      BIC LS[BIT31] TO WR[1] ;WR1 = #7FFFFFFF
U 3328, 4048.95 :12890      ADD LS[#10] TO WR[1] ;NOTE: ALU CC NOT LOADED
U 3329, 36FA.15 :12891      MOV LS[DEC.CON] TO WR[0] ;GET D D00 TO D D07
U 332A, 37FD.15 :12892      MOV LS[ALU.CC] TO WR[2] ;GET CC'S AND SAVE IN WR 2 TO CHECK LATER
U 332B, B642.95 :12893      MOV LS[#2] TO WR[1]
U 332C, C744.95 :12894      BIS LS[#4] TO WR[1] ;WR1 = #06
U 332D, 0869.3C :12895      JSR [CHECK.RESULT] ;CHECK FOR HALF CARRY L HIGH, ALU C L LOW
U 332E, 0B31.F4 :12896      JMP [LOOP.T18.9] ;LOOP ON ERROR IF ENABLED
U 332F, 3648.15 :12897
U 3330, C140.15 :12898      MOV LS[#10] TO WR[0]
U 3331, F88A.15 :12899      SUB LS[#1] FROM WR[0] ;WRO = #10 - #1 = #F
U 3332, B646.15 :12900      MCOM WR[0] TO LS[ERROR.MASK] ;ERROR.MASK = #FFFFFFF0
U 3333, 8870.5C :12901      MOV LS[#8] TO WR[0]
U 3334, 2004.15 :12902      JSR [INC.EN] ;INCREMENT ERROR NUMBER TO A
U 3335, 3640.95 :12903      MOV WR[2] TO WR[0] ;GET ALU CC BITS SAVED IN WR 2
U 3336, C744.95 :12904      MOV LS[#1] TO WR[1]
U 3337, 0869.3C :12905      BIS LS[#4] TO WR[1] ;EXPECTED DATA = BIT 0 & 2 SET
U 3338, 0B31.F4 :12906      JSR [CHECK.RESULT] ;CHECK FOR Z AND C SET
U 3339, 3648.15 :12907      JMP [LOOP.T18.9] ;LOOP ON ERROR IF ENABLED
                                LOOP.T18.11:
U 3339, 3648.15 :12908      MOV LS[#10] TO WR[0]
U 333A, C140.15 :12909      SUB LS[#1] FROM WR[0] ;WRO = #10 - #1 = #F
U 333B, F88A.15 :12910      MCOM WR[0] TO LS[ERROR.MASK] ;ERROR.MASK = #FFFFFFF0
U 333C, B646.15 :12911      MOV LS[#8] TO WR[0]
U 333D, C740.15 :12912      BIS LS[#1] TO WR[0]
U 333E, 4742.15 :12913      BIS LS[BIT1] TO WR[0]
U 333F, BE82.15 :12914      MOV WR[0] TO LS[ERROR.NUMBER] ;ERROR NUMBER = B
U 3340, 369F.15 :12915      MOV LS[ONES] TO WR[2]
U 3341, 457F.15 :12916      BIC LS[BIT31] TO WR[2] ;WR2 = #7FFFFFFF
                                ADD LS[#10] TO WR[2],
U 3342, C049.35 :12917      DT(LONG)&SET.ALU.CC
U 3343, 369E.95 :12918      MOV LS[ONES] TO WR[1] ;WR1 = #FFFFFFF
U 3344, C040.95 :12919      ADD LS[#1] TO WR[1] ;NOTE: ALU CC NOT LOADED
U 3345, B6FB.15 :12920      MOV LS[DEC.CON] TO WR[2] ;GET D D00 TO D D07 AND SAVE IN WR 2
U 3346, B7FC.15 :12921      MOV LS[ALU.CC] TO WR[0] ;GET CC'S
U 3347, B642.95 :12922      MOV LS[BIT1] TO WR[1]
U 3348, 4746.95 :12923      BIS LS[#8] TO WR[1] ;EXPECTED DATA = BIT 1 & 3 SET
U 3349, 0869.3C :12924      JSR [CHECK.RESULT] ;CHECK FOR Z AND C CLEAR
U 334A, 8B33.94 :12925      JMP [LOOP.T18.11] ;LOOP ON ERROR IF ENABLED
U 334B, 8870.5C :12926      JSR [INC.EN] ;INCREMENT ERROR NUMBER TO C
U 334C, DF26.15 :12927      MCOM LS[FFF] TO WR[0]
U 334D, 3E8A.15 :12928      MOV WR[0] TO LS[ERROR.MASK] ;ERROR MASK = #FFFFFFF0
U 334E, 2004.15 :12929      MOV WR[2] TO WR[0] ;GET D00-D07 SAVED IN WR 2
U 334F, 364A.95 :12930      MOV LS[#20] TO WR[1]
U 3350, 474C.95 :12931      BIS LS[#40] TO WR[1] ;EXPECTED DATA = #60
U 3351, 0869.3C :12932      JSR [CHECK.RESULT] ;CHECK FOR HALF CARRY L LOW, ALU C L HIGH
U 3352, 8B33.94 :12933      JMP [LOOP.T18.11] ;LOOP ON ERROR IF ENABLED
                                END.T18:
    
```


Page "TEST 19 - PSL Condition Code Tests (M8390 CPU Module)"

++

Test Description:

This test checks the PSL condition code bits. The test will check the ability to force the PSL bits via the Y bus. The PSL condition codes are output on the D BUS via the 2X4 muxes on D D00 through D D03. The other bits will be masked off and not checked during this test. The data path is as follows: WR 1 is written with a data pattern from LS. Then a MOV WR[1] TO LS[PSL.CC] is executed to load the PSL CC bits from the Y BUS through the PSL.CC PAL. The expected data is already in WR 1, so WR 0 is loaded with the PSL CC bits and the result is checked. Data patterns used are all 0's, all 1's, shifting 1's and shifting 0's. This test will also check that the PSL CC's do not change without asserting LOAD PSL CC L.

Assumptions:

It is assumed that the previous 2901 tests have run successfully.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Load WR 1 with all 0's and execute MOV WR[1] TO LS[PSL.CC] to load the PSL condition codes.
- 3) Load WR 0 with result and check.
- 4) Load WR 0 with all 1's and execute MOV WR[0] TO LS[OS]. This is to check that PSL CC's are not altered if write to LS[PSL.CC] is not executed.
- 5) Load WR 0 with result and check (should still be 0's)
- 6) Repeat steps 2, 3, 4, and 5 with complement data.
- 7) Repeat steps 2 and 3 with shifting ones starting at bit 0, until data of a 1 in bit 3 has been tested.
- 8) Repeat steps 2 and 3 with shifting zeros starting at bit 0, until data of a 0 in bit 3 has been tested.

Errors:

- Error 1 - Loading or reading of PSL CC's from Y BUS failed with data of all zeros.
- Error 2 - PSL CC's were altered on a write to OS.
- Error 3 - Loading or reading of PSL CC's from Y BUS failed with data of all ones.
- Error 4 - PSL CC's were altered on a write to OS.
- Error 5 - Loading of PSL CC's from Y BUS failed with data of shifting 1's.
- Error 6 - Loading of PSL CC's from Y BUS failed with data of shifting 0's.

Debug:

- Error 1 - This error indicates a problem with either writing the PSL CC's or reading them back through the D BUS muxes. The write occurs

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

M 6
TEST 19 - PSL Condi 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 19 - PSL Condition Code Tests (M8390 CPU Module)

Fiche 2 Frame M6

Sequence 283

Page 277

:12992 : during the MOV WR[1] TO LS[PSL.CC] u-word. During this instruction
:12993 : the following inputs to the PSL.CC PAL should be low: LOAD PSL CC L,
:12994 : BUS Y D00, BUS Y D01, BUS Y D02, and BUS Y D03. The other inputs are
:12995 : don't cares. The BUS Y inputs come from the 2901. If these are in-
:12996 : correct, suspect a broken etch. The signal LOAD PSL CC L comes from
:12997 : the decoder fed by the REG CONTROL PAL and the signal WRITE REGS L.
:12998 : If LOAD PSL CC L is incorrect, check the signal at the output of the
:12999 : decoder (pin 11). If incorrect there check the decoder inputs for
:13000 : a low on the enable (pin 15) and SEL 2 (pin 13), and a high on SEL 1
:13001 : (pin 14). If these are incorrect, suspect the REG CONTROL PAL. If
:13002 : the inputs to the PSL CC PAL are OK, then check the outputs for all
:13003 : lows. If the outputs are incorrect, then the PSL CC PAL is faulty.
:13004 : If the outputs are OK, check the PSL CC's at the 2X4 muxes. If these
:13005 : are OK then the problem is with reading the result. During the MOV
:13006 : LS[PSL.CC] TO WR instruction, check the inputs to the 2X4 MUXes for
:13007 : a low on the enable (pin 15) and SEL 2 (pin 2), and a high on SEL 1
:13008 : (pin 14). If these are incorrect, suspect the REG CONTROL PAL. If
:13009 : these signals are OK, recheck the PSL inputs. Suspect the 2X4 mux
:13010 : if the PSL inputs are OK.

:13011 :
:13012 : Error 2 - This error indicates that the PSL CC's changed on a write to
:13013 : another register. Check the signal LOAD PSL CC L at the PSL CC PAL
:13014 : during the MOV WR TO LS[OS] instruction. The signal should be high.
:13015 : If LOAD PSL CC L is high, then suspect the PSL CC PAL. If the signal
:13016 : is low, check it at the output of the decoder. The input to this
:13017 : decoder should be a high in SEL 1 and SEL 2.

:13018 :
:13019 : Error 3 - See error 1. The only difference is that BUS Y D00 through
:13020 : BUS Y D03 should be high as well as the outputs of the PSL CC PAL.
:13021 : Other signals are identical.

:13022 :
:13023 : Error 4 - see error 2

:13024 :
:13025 : Error 5 - Suspect the PSL CC PAL itself.

:13026 :
:13027 : Error 6 - Suspect the PSL CC PAL itself.

:13028 :
:13029 :
:13030 : T.19:

U 3353, B65E,15

U 3354, 3E80,15

U 3355, 10E0,15

U 3356, 8B35,64

U 3357, B640,15

U 3358, 4742,15

U 3359, 4744,15

U 335A, C746,15

U 335B, F88A,15

U 335C, 65A0,15

U 335D, 2F80,15

U 335E, 2040,15

U 335F, BE82,15

U 3360, A3C0,15

WAIT.T19.0:

MOV LS[BEGIN.TEST] TO WR[0]

MOV WR[0] TO LS[CONTROL.STATUS]

MISC [SET.CP.ATTN]

JMP [WAIT.T19.0]

MOV LS[BIT0] TO WR[0]

BIS LS[BIT1] TO WR[0]

BIS LS[BIT2] TO WR[0]

BIS LS[BIT3] TO WR[0]

MCOM WR[0] TO LS[ERROR.MASK]

CLR LS[PREVIOUS.ERROR]

CLR WR[0]

INC WR[0]

MOV WR[0] TO LS[ERROR.NUMBER]

ROL WR[0]

:SET BIT15 IN WRO FOR CONTROL AND STATUS WORD

:SET BIT15 IN CNTROL AND STATUS WORD

: BIT15 INDICATES START OF TEST TO CONSOLE

:ISSUE CPU ATTN TO CONSOLE

:LOOP HERE WAITING FOR CONSOLE TO RESPOND

:ERROR MASK = #FFFFFFF0

:CLEAR PREVIOUS ERROR NUMBER

:SET WRO TO 1

:SET ERROR NUMBER TO FIRST ERROR

:SET WRO TO 2


```

U 3361, 3E8C,15 :13047      MOV WR[0] TO LS[MODULE.NUM]      ;SET UP MODULE CODE FOR CPU MODULE
                  :13048      LOOP.T19.1:
U 3362, 2F82,95 :13049      CLR WR[1]                      ;WR1 = 0
U 3363, BEFE,95 :13050      MOV WR[1] TO LS[PSL.CC]          ;MOV DATA TO PSL CC
U 3364, B6FE,15 :13051      MOV LS[PSL.CC] TO WR[0]          ;RETRIEVE DATA
U 3365, 0869,3C :13052      JSR [CHECK.RESULT]              ;CHECK
U 3366, 0B36,24 :13053      JMP [LOOP.T19.1]                  ;LOOP ON ERROR IF ENABLED
U 3367, 8870,5C :13054      JSR [INC.EN]                      ;INCREMENT ERROR NUMBER TO 2
                  :13055      LOOP.T19.2:
U 3368, 2F82,95 :13056      CLR WR[1]                      ;WR1 = 0
U 3369, B69E,15 :13057      MOV LS[ONES] TO WR[0]            ;WRO = #FFFFFFFF
U 336A, 3EF8,15 :13058      MOV WR[0] TO LS[OS]              ;MOV DATA TO ANOTHER LOCATION
U 336B, B6FE,15 :13059      MOV LS[PSL.CC] TO WR[0]          ;RETRIEVE PSL CC
U 336C, 0869,3C :13060      JSR [CHECK.RESULT]              ;CHECK TO SEE IF PSL CC STILL CLEAR
U 336D, 0B36,84 :13061      JMP [LOOP.T19.2]                  ;LOOP ON ERROR IF ENABLED
U 336E, 8870,5C :13062      JSR [INC.EN]                      ;INCREMENT ERROR NUMBER TO 3
                  :13063      LOOP.T19.3:
U 336F, 369E,95 :13064      MOV LS[ONES] TO WR[1]            ;WR1 = #FFFFFFFF
U 3370, BEFE,95 :13065      MOV WR[1] TO LS[PSL.CC]          ;MOVE DATA TO PSL CC
U 3371, B6FE,15 :13066      MOV LS[PSL.CC] TO WR[0]          ;RETRIEVE DATA
U 3372, 0869,3C :13067      JSR [CHECK.RESULT]              ;CHECK DATA
U 3373, 8B36,F4 :13068      JMP [LOOP.T19.3]                  ;LOOP ON ERROR IF ENABLED
U 3374, 8870,5C :13069      JSR [INC.EN]                      ;INCREMENT ERROR NUMBER TO 4
                  :13070      LOOP.T19.4:
U 3375, 369E,95 :13071      MOV LS[ONES] TO WR[1]            ;WR1 = #FFFFFFFF
U 3376, 2F80,15 :13072      CLR WR[0]                      ;WRO = 0
U 3377, 3EF8,15 :13073      MOV WR[0] TO LS[OS]              ;MOVE DATA TO ANOTHER LOCATION
U 3378, B6FE,15 :13074      MOV LS[PSL.CC] TO WR[0]          ;RETRIEVE PSL CC
U 3379, 0869,3C :13075      JSR [CHECK.RESULT]              ;CHECK TO SEE IF PSL CC STILL SAME
U 337A, 0B37,54 :13076      JMP [LOOP.T19.4]                  ;LOOP ON ERROR IF ENABLED
U 337B, 8870,5C :13077      JSR [INC.EN]                      ;INCREMENT ERROR NUMBER TO 5
                  :13078      LOOP.T19.5:
U 337C, 3640,95 :13079      MOV LS[BIT0] TO WR[1]            ;WR1 = #00000001
U 337D, BEFE,95 :13080      MOV WR[1] TO LS[PSL.CC]          ;MOVE DATA TO PSL CC
U 337E, B6FE,15 :13081      MOV LS[PSL.CC] TO WR[0]          ;RETRIEVE DATA
U 337F, 0869,3C :13082      JSR [CHECK.RESULT]              ;CHECK DATA
U 3380, 0B37,C4 :13083      JMP [LOOP.T19.5]                  ;LOOP ON ERROR IF ENABLED
U 3381, B642,95 :13084      MOV LS[BIT1] TO WR[1]            ;WR1 = #00000002
U 3382, BEFE,95 :13085      MOV WR[1] TO LS[PSL.CC]          ;MOVE DATA TO PSL CC
U 3383, B6FE,15 :13086      MOV LS[PSL.CC] TO WR[0]          ;RETRIEVE DATA
U 3384, 0869,3C :13087      JSR [CHECK.RESULT]              ;CHECK DATA
U 3385, 0B37,C4 :13088      JMP [LOOP.T19.5]                  ;LOOP ON ERROR IF ENABLED
U 3386, B644,95 :13089      MOV LS[BIT2] TO WR[1]            ;WR1 = #00000004
U 3387, BEFE,95 :13090      MOV WR[1] TO LS[PSL.CC]          ;MOVE DATA TO PSL CC
U 3388, B6FE,15 :13091      MOV LS[PSL.CC] TO WR[0]          ;RETRIEVE DATA
U 3389, 0869,3C :13092      JSR [CHECK.RESULT]              ;CHECK DATA
U 338A, 0B37,C4 :13093      JMP [LOOP.T19.5]                  ;LOOP ON ERROR IF ENABLED
U 338B, 3646,95 :13094      MOV LS[BIT3] TO WR[1]            ;WR1 = #00000008
U 338C, BEFE,95 :13095      MOV WR[1] TO LS[PSL.CC]          ;MOVE DATA TO PSL CC
U 338D, B6FE,15 :13096      MOV LS[PSL.CC] TO WR[0]          ;RETRIEVE DATA
U 338E, 0869,3C :13097      JSR [CHECK.RESULT]              ;CHECK DATA
U 338F, 0B37,C4 :13098      JMP [LOOP.T19.5]                  ;LOOP ON ERROR IF ENABLED
U 3390, 8870,5C :13099      JSR [INC.EN]                      ;INCREMENT ERROR NUMBER TO 6
                  :13100      LOOP.T19.6:
U 3391, 5F40,95 :13101      MCOM LS[BIT0] TO WR[1]          ;WR1 = #FFFFFFFF
  
```

U 3392, BEFE.95	:13102	MOV WR[1] TO LS[PSL.CC]	:MOVE DATA TO PSL CC
U 3393, B6FE.15	:13103	MOV LS[PSL.CC] TO WR[0]	:RETRIEVE DATA
U 3394, 0869.3C	:13104	JSR [CHECK.RESULT]	:CHECK DATA
U 3395, 0B39.14	:13105	JMP [LOOP.T19.6]	:LOOP ON ERROR IF ENABLED
U 3396, DF42.95	:13106	MCOM LS[BIT1] TO WR[1]	:WR1 = #FFFFFFFD
U 3397, BEFE.95	:13107	MOV WR[1] TO LS[PSL.CC]	:MOVE DATA TO PSL CC
U 3398, B6FE.15	:13108	MOV LS[PSL.CC] TO WR[0]	:RETRIEVE DATA
U 3399, 0869.3C	:13109	JSR [CHECK.RESULT]	:CHECK DATA
U 339A, 0B39.14	:13110	JMP [LOOP.T19.6]	:LOOP ON ERROR IF ENABLED
U 339B, DF44.95	:13111	MCOM LS[BIT2] TO WR[1]	:WR1 = #FFFFFFFB
U 339C, BEFE.95	:13112	MOV WR[1] TO LS[PSL.CC]	:MOVE DATA TO PSL CC
U 339D, B6FE.15	:13113	MOV LS[PSL.CC] TO WR[0]	:RETRIEVE DATA
U 339E, 0869.3C	:13114	JSR [CHECK.RESULT]	:CHECK DATA
U 339F, 0B39.14	:13115	JMP [LOOP.T19.6]	:LOOP ON ERROR IF ENABLED
U 33A0, 5F46.95	:13116	MCOM LS[BIT3] TO WR[1]	:WR1 = #FFFFFFF7
U 33A1, BEFE.95	:13117	MOV WR[1] TO LS[PSL.CC]	:MOVE DATA TO PSL CC
U 33A2, B6FE.15	:13118	MOV LS[PSL.CC] TO WR[0]	:RETRIEVE DATA
U 33A3, 0869.3C	:13119	JSR [CHECK.RESULT]	:CHECK DATA
U 33A4, 0B39.14	:13120	JMP [LOOP.T19.6]	:LOOP ON ERROR IF ENABLED
	:13121	END.T19:	

Page "TEST 1A - Uniqueness test on LS addressed registers (M8390 CPU Module)"

++

Test Description:

This test runs a unique pattern on the OS reg, ALU CC reg, PSL CC reg and STAT REG to assure that they are uniquely written and read. Each of these registers uses D BUS bit 0 (among others) so only bit 0 need be checked to assure uniqueness. Each of these registers has been used before so we know that they can be written and read correctly. The test will write all the affected registers with a 0 in bit 0, then write a 1 in one register and check that only that register was affected. Then a 0 is written back to that register and another register is checked in the same manner. This test is designed to detect a short or other fault on the decoder chip which enables the Y bus to be written into these regs.

Assumptions:

It is assumed that each of the registers being checked here have been tested for data integrity by running the previous tests.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Clear WR 1 and MOV WR 1 to each register in succession.
- 3) Inc WR 1 and write to ALU CC. Read ALU CC into WR 0 and check for a 1 (only bit 0 is checked).
- 4) Clear WR 1 and read each of the other three registers for a 0 in bit 0.
- 5) Clear WR 1 and MOV WR 1 to the ALU CC reg. Now all four regs are clear again.
- 6) Repeat steps 3 through 5 using the PSL CC reg.
- 7) Repeat steps 3 through 5 using the STAT reg.
- 8) Repeat steps 3 through 5 using the OS reg.

Errors:

- Error 1 - The ALU CC reg failed to be written.
- Error 2 - Writing the ALU CC reg affected the PSL CC reg.
- Error 3 - Writing the ALU CC reg affected the STAT reg.
- Error 4 - Writing the ALU CC reg affected the OS reg.
- Error 5 - The PSL CC reg failed to be written.
- Error 6 - Writing the PSL CC reg affected the ALU CC reg.
- Error 7 - Writing the PSL CC reg affected the STAT reg.
- Error 8 - Writing the PSL CC reg affected the OS reg.
- Error 9 - The STAT reg failed to be written.
- Error A - Writing the STAT reg affected the ALU CC reg.
- Error B - Writing the STAT reg affected the PSL CC reg.
- Error C - Writing the STAT reg affected the OS reg.
- Error D - The OS reg failed to be written.
- Error E - Writing the OS reg affected the ALU CC reg.
- Error F - Writing the OS reg affected the PSL CC reg.
- Error 10 - Writing the OS reg affected the STAT reg.

```

:13177
:13178
:13179
:13180
:13181
:13182
:13183
:13184
:13185
:13186
:13187
:13188
:13189
:13190
:13191
:13192
:13193
:13194
:13195
:13196
:13197
:13198
:13199
:13200
:13201
:13202
:13203
:13204
:13205
:13206
:13207
:13208
:13209
:13210
:13211
:13212
:13213
:13214
:13215
:13216
:13217
:13218
:13219
:13220
:13221
:13222
:13223
:13224
:13225
:13226
:13227
:13228
:13229
:13230
:13231

Debug:
Errors 1,5,9,D - Any of these errors indicate a failure in logic that
was previously tested. Run previous tests again.

Errors 2,3,4,6,7,8,A,B,C,E,F,10 - Any of these errors indicates a
problem with the decoder chip which enables each of the tested reg-
isters. During the u-word that loads the reg specified in the error
report, check that only one output is low at any one time. The problem
could be a bad decoder chip or a short between the output pins.

T.1A:
MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN] ;BIT15 INDICATES START OF TEST TO CONSOLE
ISSUE CPU ATTN TO CONSOLE

WAIT.T1A.0:
JMP [WAIT.T1A.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
MOV LS[BIT0] TO WR[0]
MCOM WR[0] TO LS[ERROR.MASK] ;ERROR MASK = #FFFFFFFE
CLR LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
CLR WR[0]
INC WR[0] ;SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
ROL WR[0] ;SET WRO TO 2
MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
CLR WR[1]
MOV WR[1] TO LS[ALU.CC]
MOV WR[1] TO LS[PSL.CC]
MOV WR[1] TO LS[MDT]

;NOTE: THE STAT REGISTER CONTAINS THE
;REGISTERS MDT, SIZE, AND INTER-
;RUPT.VECTOR.

MOV WR[1] TO LS[OS]
LOOP.T1A.1:
MOV LS[BIT0] TO WR[1] ;WR1 = #00000001
MOV WR[1] TO LS[ERROR.NUMBER] ;ERROR NUMBER = 1
MOV WR[1] TO LS[ALU.CC] ;MOVE BIT TO ALU.CC
MOV LS[ALU.CC] TO WR[0] ;RETRIEVE
JSR [CHECK.RESULT] ;CHECK TO BE SURE IT WAS WRITTEN
JMP [LOOP.T1A.1] ;LOOP ON ERROR IF ENABLED
JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
CLR WR[1] ;EXPECTED DATA FOR NEXT THREE SECTIONS = 0

LOOP.T1A.2:
MOV LS[PSL.CC] TO WR[0] ;CHECK THAT PSL.CC IS STILL CLEAR
JSR [CHECK.RESULT] ;LOOP ON ERROR IF ENABLED
JMP [LOOP.T1A.1] ;INCREMNT ERROR NUMBER TO 3
JSR [INC.EN]

LOOP.T1A.3:
MOV LS[MDT] TO WR[0] ;CHECK THAT MDT IS STILL CLEAR
JSR [CHECK.RESULT] ;LOOP ON ERROR IF ENABLED
JMP [LOOP.T1A.1] ;INCREMNT ERROR NUMBER TO 4
JSR [INC.EN]

LOOP.T1A.4:

```

U 33A5, B65E,15
U 33A6, 3E80,15
U 33A7, 10E0,15
U 33A8, 0B3A,84
U 33A9, B640,15
U 33AA, F88A,15
U 33AB, 65A0,15
U 33AC, 2F80,15
U 33AD, 2040,15
U 33AE, BE82,15
U 33AF, A3C0,15
U 33B0, 3E8C,15
U 33B1, 2F82,95
U 33B2, BFFC,95
U 33B3, BEFE,95
U 33B4, 3EFC,95
U 33B5, BEF8,95
U 33B6, 3640,95
U 33B7, 3E82,95
U 33B8, BFFC,95
U 33B9, B7FC,15
U 33BA, 0B69,3C
U 33BB, 0B3B,64
U 33BC, 8870,5C
U 33BD, 2F82,95
U 33BE, B6FE,15
U 33BF, 0B69,3C
U 33C0, 0B3B,64
U 33C1, 8870,5C
U 33C2, 36FC,15
U 33C3, 0B69,3C
U 33C4, 0B3B,64
U 33C5, 8870,5C


```

U 33C6, B6F8,15 :13232      MOV LS[OS] TO WR[0]
U 33C7, 0869,3C :13233      JSR [CHECK.RESULT]      ;CHECK THAT OS IS STILL CLEAR
U 33C8, 0B3B,64 :13234      JMP [LOOP.T1A.1]      ;LOOP ON ERROR IF ENABLED
U 33C9, 2F82,95 :13235      CLR WR[1]
U 33CA, BFFC,95 :13236      MOV WR[1] TO LS[ALU.CC]      ;RESET ALU.CC
                                LOOP.T1A.5:
U 33CB, B644,95 :13238      MOV LS[#4] TO WR[1]
U 33CC, 2042,95 :13239      INC WR[1]
U 33CD, 3E82,95 :13240      MOV WR[1] TO LS[ERROR.NUMBER]      ;ERROR NUMBER = 5
U 33CE, 3640,95 :13241      MOV LS[BIT0] TO WR[1]      ;WR1 = #00000001
U 33CF, BEFE,95 :13242      MOV WR[1] TO LS[PSL.CC]      ;MOVE BIT TO PSL.CC
U 33D0, B7FC,15 :13243      MOV LS[ALU.CC] TO WR[0]
U 33D1, 3E10,15 :13244      MOV WR[0] TO LS[T8]
U 33D2, B6FE,15 :13245      MOV LS[PSL.CC] TO WR[0]
U 33D3, 0869,3C :13246      JSR [CHECK.RESULT]      ;CHECK TO BE SURE IT WAS WRITTEN
U 33D4, 0B3C,B4 :13247      JMP [LOOP.T1A.5]      ;LOOP ON ERROR IF ENABLED
U 33D5, 8870,5C :13248      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 6
U 33D6, 2F82,95 :13249      CLR WR[1]      ;EXPECTED DATA FOR NEXT THREE SECTIONS = 0
                                LOOP.T1A.6:
U 33D7, B610,15 :13251      MOV LS[T8] TO WR[0]
U 33D8, 0869,3C :13252      JSR [CHECK.RESULT]      ;CHECK THAT ALU.CC IS STILL CLEAR
U 33D9, 0B3C,B4 :13253      JMP [LOOP.T1A.5]      ;LOOP ON ERROR IF ENABLED
U 33DA, 8870,5C :13254      JSR [INC.EN]      ;INCREMNT ERROR NUMBER TO 7
                                LOOP.T1A.7:
U 33DB, 36FC,15 :13256      MOV LS[MDT] TO WR[0]
U 33DC, 0869,3C :13257      JSR [CHECK.RESULT]      ;CHECK THAT MDT IS STILL CLEAR
U 33DD, 0B3C,B4 :13258      JMP [LOOP.T1A.5]      ;LOOP ON ERROR IF ENABLED
U 33DE, 8870,5C :13259      JSR [INC.EN]      ;INCREMNT ERROR NUMBER TO 8
                                LOOP.T1A.8:
U 33DF, B6F8,15 :13261      MOV LS[OS] TO WR[0]
U 33E0, 0869,3C :13262      JSR [CHECK.RESULT]      ;CHECK THAT OS IS STILL CLEAR
U 33E1, 0B3C,B4 :13263      JMP [LOOP.T1A.5]      ;LOOP ON ERROR IF ENABLED
U 33E2, 2F82,95 :13264      CLR WR[1]
U 33E3, BEFE,95 :13265      MOV WR[1] TO LS[PSL.CC]      ;RESET PSL.CC
U 33E4, BFFC,95 :13266      MOV WR[1] TO LS[ALU.CC]      ;RESET ALU.CC
                                LOOP.T1A.9:
U 33E5, 3646,95 :13268      MOV LS[#8] TO WR[1]
U 33E6, 2042,95 :13269      INC WR[1]
U 33E7, 3E82,95 :13270      MOV WR[1] TO LS[ERROR.NUMBER]      ;ERROR NUMBER = 9
U 33E8, 3640,95 :13271      MOV LS[BIT0] TO WR[1]      ;WR1 = #00000001
U 33E9, 3EFC,95 :13272      MOV WR[1] TO LS[MDT]      ;MOVE BIT TO MDT
U 33EA, B7FC,15 :13273      MOV LS[ALU.CC] TO WR[0]
U 33EB, 3E10,15 :13274      MOV WR[0] TO LS[T8]
U 33EC, 36FC,15 :13275      MOV LS[MDT] TO WR[0]
U 33ED, 0869,3C :13276      JSR [CHECK.RESULT]      ;CHECK TO BE SURE IT WAS WRITTEN
U 33EE, 0B3E,54 :13277      JMP [LOOP.T1A.9]      ;LOOP ON ERROR IF ENABLED
U 33EF, 8870,5C :13278      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO A
U 33F0, 2F82,95 :13279      CLR WR[1]      ;EXPECTED DATA FOR NEXT THREE SECTIONS = 0
                                LOOP.T1A.A:
U 33F1, B610,15 :13281      MOV LS[T8] TO WR[0]
U 33F2, 0869,3C :13282      JSR [CHECK.RESULT]      ;CHECK THAT ALU.CC IS STILL CLEAR
U 33F3, 0B3E,54 :13283      JMP [LOOP.T1A.9]      ;LOOP ON ERROR IF ENABLED
U 33F4, 8870,5C :13284      JSR [INC.EN]      ;INCREMNT ERROR NUMBER TO B
                                LOOP.T1A.B:
U 33F5, B6FE,15 :13286      MOV LS[PSL.CC] TO WR[0]
    
```

```

U 33F6, 0869,3C :13287      JSR [CHECK.RESULT]      ;CHECK THAT PSL.CC IS STILL CLEAR
U 33F7, 0B3E,54 :13288      JMP [LOOP.T1A.9]      ;LOOP ON ERROR IF ENABLED
U 33F8, 8870,5C :13289      JSR [INC.EN]      ;INCREMNT ERROR NUMBER TO C
:13290
U 33F9, B6F8,15 :13291      LOOP.T1A.C:      MOV LS[OS] TO WR[0]
U 33FA, 0869,3C :13292      JSR [CHECK.RESULT]      ;CHECK THAT OS IS STILL CLEAR
U 33FB, 0B3E,54 :13293      JMP [LOOP.T1A.9]      ;LOOP ON ERROR IF ENABLED
U 33FC, 2F82,95 :13294      CLR WR[1]
U 33FD, 3EFC,95 :13295      MOV WR[1] TO LS[MDT]      ;RESET MDT
U 33FE, BFFC,95 :13296      MOV WR[1] TO LS[ALU.CC]      ;RESET ALU.CC
:13297
U 33FF, 3646,95 :13298      LOOP.T1A.D:      MOV LS[#8] TO WR[1]
U 3400, C744,95 :13299      BIS LS[#4] TO WR[1]
U 3401, 2042,95 :13300      INC WR[1]
U 3402, 3E82,95 :13301      MOV WR[1] TO LS[ERROR.NUMBER] ;ERROR NUMBER = D
U 3403, 3640,95 :13302      MOV LS[BIT0] TO WR[1]      ;WR1 = #00000001
U 3404, BEF8,95 :13303      MOV WR[1] TO LS[OS]      ;MOVE BIT TO OS
U 3405, B7FC,15 :13304      MOV LS[ALU.CC] TO WR[0]
U 3406, 3E10,15 :13305      MOV WR[0] TO LS[T8]      ;SAVE ALU BITS
U 3407, B6F8,15 :13306      MOV LS[OS] TO WR[0]      ;RETRIEVE
U 3408, 0869,3C :13307      JSR [CHECK.RESULT]      ;CHECK TO BE SURE IT WAS WRITTEN
U 3409, 8B3F,F4 :13308      JMP [LOOP.T1A.D]      ;LOOP ON ERROR IF ENABLED
U 340A, 8870,5C :13309      JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO E
U 340B, 2F82,95 :13310      CLR WR[1]      ;EXPECTED DATA FOR NEXT THREE SECTIONS = 0
:13311
U 340C, B610,15 :13312      LOOP.T1A.E:      MOV LS[T8] TO WR[0]
U 340D, 0869,3C :13313      JSR [CHECK.RESULT]      ;CHECK THAT ALU.CC IS STILL CLEAR
U 340E, 8B3F,F4 :13314      JMP [LOOP.T1A.D]      ;LOOP ON ERROR IF ENABLED
U 340F, 8870,5C :13315      JSR [INC.EN]      ;INCREMNT ERROR NUMBER TO F
:13316
U 3410, B6FE,15 :13317      LOOP.T1A.F:      MOV LS[PSL.CC] TO WR[0]
U 3411, 0869,3C :13318      JSR [CHECK.RESULT]      ;CHECK THAT PSL.CC IS STILL CLEAR
U 3412, 8B3F,F4 :13319      JMP [LOOP.T1A.D]      ;LOOP ON ERROR IF ENABLED
U 3413, 8870,5C :13320      JSR [INC.EN]      ;INCREMNT ERROR NUMBER TO 10
:13321
U 3414, 36FC,15 :13322      LOOP.T1A.10:     MOV LS[MDT] TO WR[0]
U 3415, 0869,3C :13323      JSR [CHECK.RESULT]      ;CHECK THAT MDT IS STILL CLEAR
U 3416, 8B3F,F4 :13324      JMP [LOOP.T1A.D]      ;LOOP ON ERROR IF ENABLED
U 3417, 2F82,95 :13325      CLR WR[1]
U 3418, BEF8,95 :13326      MOV WR[1] TO LS[OS]      ;RESET OS
:13327      END.T1A:
  
```


:13328
:13329
:13330
:13331
:13332
:13333
:13334
:13335
:13336
:13337
:13338
:13339
:13340
:13341
:13342
:13343
:13344
:13345
:13346
:13347
:13348
:13349
:13350
:13351
:13352
:13353
:13354
:13355
:13356
:13357
:13358
:13359
:13360
:13361
:13362
:13363
:13364
:13365
:13366
:13367
:13368
:13369
:13370
:13371
:13372
:13373
:13374
:13375
:13376
:13377
:13378
:13379
:13380
:13381
:13382

Page "TEST 1B - Byte and Word Write to LS (M8390 CPU Module)"

++

Test Description:

This test checks the logic which enables the write to LS for byte and word data. The SIZE REG is used to specify the data type. The data path is as follows: The SIZE REG is loaded with the code for byte, word or longword data and a temporary LS location is written with known longword data. Then a MOV WR to LS is issued with the CC code = 2. The CC CONTROL PAL sets up DATA TYPE 0 and DATA TYPE 1 according to the SIZE REG. The LS CONTROL PAL uses DATA TYPE 0 and DATA TYPE 1 to control the write enable to LS. After the new data is written in LS, the LS location is checked to assure that only the byte or word portion was written, while the rest of the bits were unchanged. A test is also run to check that setting COMPAT MODE in the PSL will force a word write, even if the data type indicates a longword is to be written. Data patterns used are all ones and all zeros.

Assumptions:

It is assumed that the previous condition code tests have run successfully.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Set up the SIZE REG for byte data (SIZE REG=0).
- 3) Write a temporary LS location with all zeros. Then load WR 1 with the expected data.
- 4) Load WR 0 with all 1's and execute a MOV WR[0] TO LS to the temporary LS location.
- 5) Load WR 0 from the LS location and check for FF (only a byte should have been written).
- 6) Set up the SIZE REG for word data (SIZE REG=1) and repeat steps 3 through 5, except check for FFFF (only a word should have been written).
- 7) Load PSL with bit 31 set (compat mode).
- 8) Repeat steps 3 and 4 except the CC field will be 0 (forcing a longword data type). Load WR 0 from the LS location and check for FFFF (COMPAT mode should have forced a word write).
- 9) Clear COMPAT mode in the PSL and repeat step 8. The expected data should now be FFFFFFFF (longword write).

Errors:

- Error 1 - Write byte to LS failed.
- Error 2 - Write word to LS failed (using SIZE REG).
- Error 3 - Write word to LS failed (using COMPAT MODE).
- Error 4 - COMPAT MODE failed to clear.

Debug:

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

MICRO2 1L(03) TEST 1B - Byte and Word Write to LS (M8390 CPU Module)
H 7
3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module

Fiche 2 Frame H7

Sequence 291
Rev 01.00 Page 285

:13383 : Error 1 - This error indicates a problem with the LS CONTROL PAL or
:13384 : the ENABLE logic it feeds. First check the outputs from the LS
:13385 : CONTROL PAL during the MOV instruction which writes a byte in LS.
:13386 : Pins 13 and 14 should be high, while pin 12 should be low. If these
:13387 : are correct, check the 'OR' gates they feed. ENABLE LONG H and ENABLE
:13388 : WORD H should both be low. If they are, there is a possibility that
:13389 : they are not getting to the LS RAM chips. A float at the enable (pin
:13390 : 17) of the RAMS will allow the write to always occur. If the output
:13391 : of the LS CONTROL PAL is in error, check the inputs signals DATA TYPE
:13392 : 0 H and DATA TYPE 1 H for a low. If these two inputs are OK, the PAL
:13393 : is probably bad. These 2 signals have been used internally before in
:13394 : the CC CONTROL PAL, but it is possible that they fail to get to the
:13395 : output of the PAL or that the etch is broken. If either of these in-
:13396 : puts is high, check them at the CC CONTROL PAL.

:13397 :
:13398 : Error 2 - If this is the first error, the most likely problem is with
:13399 : the LS CONTROL PAL itself.

:13400 :
:13401 : Error 3 - If this is the first error, the most likely problem is with
:13402 : the CC CONTROL PAL or the PSL COMPAT MODE bit. During the MOV WR TO
:13403 : LS instruction, check the COMPAT MODE H input to the CC CONTROL PAL.
:13404 : This input should be high making the output DATA TYPE 1 H low. If the
:13405 : input is OK but the output is high, suspect the CC CONTROL PAL itself.
:13406 : If COMPAT MODE H is low, check the output at the CM.RDEST PAL. If
:13407 : COMPAT MODE H is low here, check the inputs during the MOV instruction
:13408 : which writes the PSL for a low on LOAD PSL L and a high on BUS Y D31.
:13409 : If these inputs are correct, the PAL is bad.

:13410 :
:13411 : Error 4 - This error indicates a problem with the CM.RDEST PAL. Check
:13412 : the inputs for a low on LOAD PSL L and a low on BUS Y D31 during the
:13413 : MOV instruction that loads the PSL. If these are correct, and COMPAT
:13414 : MODE H fails to go low, the PAL itself is bad.

:13415 :
:13416 :
:13417 : T.1B:

U 3419, B65E,15 :13418 : MOV LS[BEGIN.TEST] TO WR[0] :SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
U 341A, 3E80,15 :13419 : MOV WR[0] TO LS[CONTROL.STATUS] :SET BIT15 IN CNTROL AND STATUS WORD
:13420 : : BIT15 INDICATES START OF TEST TO CONSOLE
U 341B, 10E0,15 :13421 : MISC [SET.CP.ATTN] :ISSUE CPU ATTN TO CONSOLE
:13422 :
WAIT.T1B.0: :13423 : JMP [WAIT.T1B.0] :LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 341C, 8B41,C4 :13424 : CLR LS[ERROR.MASK] :CLEAR ERROR MASK.
U 341D, E58A,15 :13425 : CLR LS[PREVIOUS.ERROR] :CLEAR PREVIOUS ERROR NUMBER
U 341E, 65A0,15 :13426 : CLR WR[0]
U 341F, 2F80,15 :13427 : INC WR[0] :SET WRO TO 1
U 3420, 2040,15 :13428 : MOV WR[0] TO LS[ERROR.NUMBER] :SET ERROR NUMBER TO FIRST ERROR
U 3421, BE82,15 :13429 : ROL WR[0] :SET WRO TO 2
U 3422, A3C0,15 :13430 : MOV WR[0] TO LS[MODULE.NUM] :SET UP MODULE CODE FOR CPU MODULE
U 3423, 3E8C,15 :13431 :
:13432 :
LOOP.T1B.1: :13433 : CLR LS[SIZE] :SIZE = BYTE (00B)
U 3424, 65FC,15 :13434 : CLR LS[T8] :CLEAR THE LS LOCATION TO BE USED
U 3425, E510,15 :13435 : MOV LS[0FF] TO WR[1] :EXPECTED DATA = #000000FF
U 3426, 3626,95 :13436 : MOV LS[ONES] TO WR[0] :WRO = #FFFFFFFF
U 3427, B69E,15 :13437 : MOV WR[0] TO LS[T8].


```

U 3428, BE10,55 :13438      DT(SIZE)&SET.ALU.CC      ;WRITE TO LS USING SIZE
U 3429, B610,15 :13439      MOV LS[T8] TO WR[0]      ;RETRIEVE WRITTEN DATA
U 342A, 0869,3C :13440      JSR [CHECK.RESULT]      ;CHECK THAT ONLY LOWER BYTE WAS WRITTEN
U 342B, 0B42,44 :13441      JMP [LOOP.T1B.1]        ;LOOP ON ERROR IF ENABLED
U 342C, 8870,5C :13442      JSR [INC.EN]            ;INCREMENT ERROR NUMBER TO 2
                                :13443
                                LOOP.T1B.2:
U 342D, 3640,95 :13444      MOV LS[BIT0] TO WR[1]
U 342E, 3EFC,95 :13445      MOV WR[1] TO LS[SIZE]
U 342F, E510,15 :13446      CLR LS[T8]
U 3430, B628,95 :13447      MOV LS[ONES] TO WR[0]
U 3431, B69E,15 :13448      MOV LS[ONES] TO WR[0]
                                :13449
                                MOV WR[0] TO LS[T8],
U 3432, BE10,55 :13450      DT(SIZE)&SET.ALU.CC      ;WRITE TO LS USING SIZE
U 3433, B610,15 :13451      MOV LS[T8] TO WR[0]      ;RETRIEVE WRITTEN DATA
U 3434, 0869,3C :13452      JSR [CHECK.RESULT]      ;CHECK THAT ONLY LOWER WORD WAS WRITTEN
U 3435, 0B42,D4 :13453      JMP [LOOP.T1B.2]        ;LOOP ON ERROR IF ENABLED
U 3436, 8870,5C :13454      JSR [INC.EN]            ;INCREMENT ERROR NUMBER TO 3
                                :13455
                                LOOP.T1B.3:
U 3437, B67E,95 :13456      MOV LS[BIT31] TO WR[1]
U 3438, 3FFE,95 :13457      MOV WR[1] TO LS[PSL.HW]
U 3439, E510,15 :13458      CLR LS[T8]
U 343A, B628,95 :13459      MOV LS[ONES] TO WR[1]
U 343B, B69E,15 :13460      MOV LS[ONES] TO WR[0]
                                :13461
                                MOV WR[0] TO LS[T8],
U 343C, BE10,35 :13462      DT(LONG)&SET.ALU.CC      ;WRITE TO LS USING LONGWORD DATA TYPE
U 343D, B610,15 :13463      MOV LS[T8] TO WR[0]      ;RETRIEVE WRITTEN DATA
U 343E, 0869,3C :13464      JSR [CHECK.RESULT]      ;CHECK THAT ONLY LOWER WORD WAS WRITTEN
U 343F, 8B43,74 :13465      JMP [LOOP.T1B.3]        ;LOOP ON ERROR IF ENABLED
U 3440, 8870,5C :13466      JSR [INC.EN]            ;INCREMENT ERROR NUMBER TO 4
                                :13467
                                LOOP.T1B.4:
U 3441, 2F80,15 :13468      CLR WR[0]
U 3442, BFFE,15 :13469      MOV WR[0] TO LS[PSL.HW]
U 3443, E510,15 :13470      CLR LS[T8]
U 3444, 369E,95 :13471      MOV LS[ONES] TO WR[1]
U 3445, B69E,15 :13472      MOV LS[ONES] TO WR[0]
                                :13473
                                MOV WR[0] TO LS[T8],
U 3446, BE10,35 :13474      DT(LONG)&SET.ALU.CC      ;WRITE TO LS USING LONGWORD DATA TYPE
U 3447, B610,15 :13475      MOV LS[T8] TO WR[0]      ;RETRIEVE WRITTEN DATA
U 3448, 0869,3C :13476      JSR [CHECK.RESULT]      ;CHECK THAT ONLY LOWER WORD WAS WRITTEN
U 3449, 0B44,14 :13477      JMP [LOOP.T1B.4]        ;LOOP ON ERROR IF ENABLED
                                :13478
                                END.T1B:
  
```

Page "TEST 1C - BUS D D07 and BUS D D06 Test (M8390 cPU Module)"

Test Description:

This test checks the D06 and D07 outputs of the DATA TYPE CTL PAL. In so doing, the MDT CTL lines from the LS CONTROL PAL will be tested as well as the data type signals generated by the MEM.REQ instruction. Memory references will not be enabled for this test so no MEMORY REQUEST H will be generated. The data path is as follows: A MEM.REQ instruction is executed with one of the four possible DT modes. The CC CONTROL PAL generates DATA TYPE 0 and 1 and sends it to the LS CONTROL PAL. The LS CONTROL PAL generates MDT CTL 0 and 1 and sends them to the DATA TYPE CTL PAL. At the end of the cycle, BUS D D06 and BUS D D07 are latched. The data is checked by reading the status reg and checking bits 6 and 7. The other bits will be masked out.

Assumptions:

It is assumed that all previous tests have run successfully.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Execute a MEM.REQ with DT=11(B) for longword data type.
- 3) Execute a MOV LS[7E] TO WR[0] and check bits 7 and 6 for a 11(B) respectively.
- 4) Repeat step 3 to assure that data does not change.
- 5) Execute a MEM.REQ with DT=01(B) for word data type.
- 6) Execute a MOV LS[7E] TO WR[0] and check bits 7 and 6 for a 01(B) respectively.
- 7) Repeat step 6 to assure that data does not change.
- 8) Execute a MEM.REQ with DT=00(B) for byte data type.
- 9) Execute a MOV LS[7E] TO WR[0] and check bits 7 and 6 for a 00(B) respectively.
- 10) Repeat step 9 to assure that data does not change.
- 11) Load SIZE REG with 11 for longword data.
- 12) Execute a MEM.REQ with DT=10(B) for size reg data type.
- 13) Execute a MOV LS[7E] TO WR[0] and check bits 7 and 6 for a 11(B) respectively.
- 14) Load SIZE REG with 01 for word data.
- 15) Execute a MEM.REQ with DT=10(B) for size reg data type.
- 16) Execute a MOV LS[7E] TO WR[0] and check bits 7 and 6 for a 01(B) respectively.
- 17) Load SIZE REG with 00 for byte data.
- 18) Execute a MEM.REQ with DT=10(B) for size reg data type.
- 19) Execute a MOV LS[7E] TO WR[0] and check bits 7 and 6 for a 00(B) respectively.

Errors:

- Error 1 - MEM.REQ with longword data type failed to load BUS D D07 with a 1 and BUS D D06 with a 1.
Error 2 - MOVE altered BUS D D07 or BUS D D06.

:13534 : Error 3 - MEM.REQ with word data type failed to load BUS D D07 with
:13535 : a 0 and BUS D D06 with a 1.
:13536 : Error 4 - MOVE altered BUS D D07 or BUS D D06.
:13537 : Error 5 - MEM.REQ with byte data type failed to load BUS D D07 with
:13538 : a 0 and BUS D D06 with a 0.
:13539 : Error 6 - MOVE altered BUS D D07 or BUS D D06.
:13540 : Error 7 - MEM.REQ failed to set DATA TYPE 0 or DATA TYPE 1 correctly
:13541 : with SIZE REG data type in DT field.
:13542 : Error 8 - MEM.REQ failed to set DATA TYPE 0 or DATA TYPE 1 correctly
:13543 : with SIZE REG data type in DT field.
:13544 : Error 9 - MEM.REQ failed to set DATA TYPE 0 or DATA TYPE 1 correctly
:13545 : with SIZE REG data type in DT field.
:13546 :
:13547 : Debug:
:13548 :
:13549 : Error 1 - This error can be caused by several problems. First check
:13550 : the CC CONTROL PAL outputs DATA TYPE 1 H and DATA TYPE 0 H. They
:13551 : both be high during the MEM.REQ instruction. If not check the inputs
:13552 : for the following: CSR 22-19 should equal 0011(8) respectively, and
:13553 : CSR 06 and CSR 05 should both be high. If the inputs are OK, suspect
:13554 : the PAL if both outputs are not high. If DATA TYPE 1 and 0 are OK,
:13555 : check the outputs MDT CTL 1 H and MDT CTL 0 H at the LS CONTROL PAL.
:13556 : MDT CTL 1 H should be high and MDT CTL 0 H should be low. Check the
:13557 : inputs for a 0011(8) on CSR 22 - CSR 19. The other inputs have been
:13558 : checked previously. If the output is incorrect but the inputs are OK,
:13559 : suspect the LS CONTROL PAL itself. If MDT CTL is OK, then check BUS D
:13560 : D07 H and BUS D D06 H at the output of the DATA TYPE CTL PAL. These
:13561 : become latched at the end of the MEM.REQ instruction and are enabled by
:13562 : the MOV LS[7E] TO WR instruction. Both outputs should be high during
:13563 : the MOV instruction. If not check the enable on pin 11 for a low. If
:13564 : the enable is OK, then the inputs will have to be checked during the
:13565 : MEM.REQ instruction. MDT CTL 1 H should be high and MDT CTL 0 H should
:13566 : be low. If these are OK, the PAL is probably bad.
:13567 :
:13568 : Error 2 - If this is the first error, check that MDT CTL 1 H and MDT
:13569 : CTL 0 H are both high during the MOV instruction. If they are, the
:13570 : DATA TYPE CTL PAL is suspect. If they are incorrect, the LS CONTROL
:13571 : PAL is suspect.
:13572 :
:13573 : Error 3 - Same as error 1 except for the following signals: DATA TYPE
:13574 : 1 H should be low, CSR 06 H should be low, MDT CTL 1 H should be low,
:13575 : MDT CTL 0 H should be high, and BUS D D07 H should be low.
:13576 :
:13577 : Error 4 - Same as error 2.
:13578 :
:13579 : Error 5 - Same as error 1 except for the following signals: DATA TYPE
:13580 : 1 H and DATA TYPE 0 H should both be low, CSR 06 H and CSR 05 H should
:13581 : both be low, MDT CTL 0 should be low (both MDT CTL signals are low),
:13582 : and both BUS D D07 H and BUS D D06 H should be low.
:13583 :
:13584 : Error 6 - Same as error 2.
:13585 :
:13586 : Error 7 to 9 - The most likely suspect is the CC CONTROL PAL itself.
:13587 : All logic after this PAL is identical to previous errors.
:13588 :

```

:13589
:13590 T.1C:
U 344A, B65E,15 :13591 MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
U 344B, 3E80,15 :13592 MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
:13593 ;BIT15 INDICATES START OF TEST TO CONSOLE
U 344C, 10E0,15 :13594 MISC [SET.CP.ATTN] ;ISSUE CPU ATTN TO CONSOLE
:13595 WAIT.T1C.0:
U 344D, 0B44,D4 :13596 JMP [WAIT.T1C.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 344E, 3660,15 :13597 MOV LS[INTERUPT.EN] TO WR[0]
U 344F, 476C,15 :13598 BIS LS[DISABLE.MEM.REF] TO WR[0]
U 3450, 3E80,15 :13599 MOV WR[0] TO LS[CONTROL.STATUS] ;DISABLE MEMORY REFERNECES
U 3451, 10E0,15 :13600 MISC [SET.CP.ATTN] ;CALL CONSOLE
:13601 WAIT.T1C.00:
U 3452, 8B45,24 :13602 JMP [WAIT.T1C.00] ;WAIT FOR CONSOLE
U 3453, B69E,15 :13603 MOV LS[ONES] TO WR[0]
U 3454, 454C,15 :13604 BIC LS[BIT6] TO WR[0]
U 3455, C54E,15 :13605 BIC LS[BIT7] TO WR[0]
U 3456, 3E8A,15 :13606 MOV WR[0] TO LS[ERROR.MASK] ;ERROR MASK = #FFFF3FFF
U 3457, 65A0,15 :13607 CLR LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
U 3458, 2F80,15 :13608 CLR WR[0]
U 3459, 2040,15 :13609 INC WR[0] ;SET WRO TO 1
U 345A, BE82,15 :13610 MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
U 345B, A3C0,15 :13611 ROL WR[0] ;SET WRO TO 2
U 345C, 3E8C,15 :13612 MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
:13613 LOOP.T1C.1:
U 345D, 3640,95 :13614 MOV LS[#1] TO WR[1]
U 345E, 3E82,95 :13615 MOV WR[1] TO LS[ERROR.NUMBER] ;ERROR NUMBER = 1
U 345F, 1D0B,F5 :13616 MEM.REQ[WRITE.CSR] ADRS[T5.SUB] DT[LONG] ;ISSUE MEM.REQ WITH LONGWORD DATA TYPE
U 3460, 36FC,15 :13617 MOV LS[MDT] TO WR[0]
U 3461, 364C,95 :13618 MOV LS[BIT6] TO WR[1]
U 3462, C74E,95 :13619 BIS LS[BIT7] TO WR[1] ;EXPECTED DATA = #0000C000
U 3463, 0869,3C :13620 JSR [CHECK.RESULT] ;CHECK FOR BIT 6 & 7 SET
U 3464, 8B45,D4 :13621 JMP [LOOP.T1C.1] ;LOOP ON ERROR IF ENABLED
U 3465, 8870,5C :13622 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
:13623 T1C.2:
U 3466, 36FC,15 :13624 MOV LS[MDT] TO WR[0]
U 3467, 364C,95 :13625 MOV LS[BIT6] TO WR[1]
U 3468, C74E,95 :13626 BIS LS[BIT7] TO WR[1] ;EXPECTED DATA = #0000C000
U 3469, 0869,3C :13627 JSR [CHECK.RESULT] ;CHECK FOR BIT 6 & 7 STILL SET
U 346A, 8B45,D4 :13628 JMP [LOOP.T1C.1] ;LOOP ON ERROR IF ENABLED
:13629 LOOP.T1C.3:
U 346B, B642,95 :13630 MOV LS[#2] TO WR[1]
U 346C, 2042,95 :13631 INC WR[1]
U 346D, 3E82,95 :13632 MOV WR[1] TO LS[ERROR.NUMBER] ;ERROR NUMBER = 3
U 346E, 9D0B,B5 :13633 MEM.REQ[WRITE.CSR] ADRS[T5.SUB] DT[WORD] ;ISSUE MEM.REQ WITH WORD DATA TYPE
U 346F, 36FC,15 :13634 MOV LS[MDT] TO WR[0]
U 3470, 364C,95 :13635 MOV LS[BIT6] TO WR[1] ;EXPECTED DATA = #00004000
U 3471, 0869,3C :13636 JSR [CHECK.RESULT] ;CHECK FOR BIT 6 SET
U 3472, 8B46,B4 :13637 JMP [LOOP.T1C.3] ;LOOP ON ERROR IF ENABLED
U 3473, 8870,5C :13638 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 4
:13639 T1C.4:
U 3474, 36FC,15 :13640 MOV LS[MDT] TO WR[0]
U 3475, 364C,95 :13641 MOV LS[BIT6] TO WR[1] ;EXPECTED DATA = #00004000
U 3476, 0869,3C :13642 JSR [CHECK.RESULT] ;CHECK FOR BIT 6 STILL SET
U 3477, 8B46,B4 :13643 JMP [LOOP.T1C.3] ;LOOP ON ERROR IF ENABLED

```



```

:13644 LOOP.T1C.5:
U 3478. B644.95 :13645 MOV LS[#4] TO WR[1]
U 3479. 2042.95 :13646 INC WR[1]
U 347A. 3E82.95 :13647 MOV WR[1] TO LS[ERROR.NUMBER] ;ERROR NUMBER = 5
U 347B. 1D0B.95 :13648 MEM.REQ[WRITE.CSR] ADRS[T5.SUB] DT[BYTE] ;ISSUE MEM.REQ WITH BYTE DATA TYPE
U 347C. 36FC.15 :13649 MOV LS[MDT] TO WR[0]
U 347D. 2F82.95 :13650 CLR WR[1] ;EXPECTED DATA = #00000000
U 347E. 0869.3C :13651 JSR [CHECK.RESULT] ;CHECK FOR BIT 6 & 7 CLEAR
U 347F. 0B47.84 :13652 JMP [LOOP.T1C.5] ;LOOP ON ERROR IF ENABLED
U 3480. 8870.5C :13653 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 6
:13654
U 3481. 36FC.15 :13655 T1C.6: MOV LS[MDT] TO WR[0]
U 3482. 2F82.95 :13656 CLR WR[1] ;EXPECTED DATA = #00000000
U 3483. 0869.3C :13657 JSR [CHECK.RESULT] ;CHECK FOR BIT 6 & 7 STILL CLEAR
U 3484. 0B47.84 :13658 JMP [LOOP.T1C.5] ;LOOP ON ERROR IF ENABLED
U 3485. 8870.5C :13659 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 7
:13660
U 3486. 3640.95 :13661 LOOP.T1C.7: MOV LS[BIT0] TO WR[1]
U 3487. 2042.95 :13662 INC WR[1]
U 3488. 3EFC.95 :13663 MOV WR[1] TO LS[SIZE] ;SIZE = LONGWORD (11B)
U 3489. 9D0B.D5 :13664 MEM.REQ[WRITE.CSR] ADRS[T5.SUB] DT[SIZE] ;ISSUE MEM.REQ WITH SIZE DATA TYPE
U 348A. 36FC.15 :13665 MOV LS[MDT] TO WR[0]
U 348B. 364C.95 :13666 MOV LS[BIT6] TO WR[1]
U 348C. C74E.95 :13667 BIS LS[BIT7] TO WR[1] ;EXPECTED DATA = #0000C000
U 348D. 0869.3C :13668 JSR [CHECK.RESULT] ;CHECK FOR BIT 6 & 7 SET
U 348E. 8B48.64 :13669 JMP [LOOP.T1C.7] ;LOOP ON ERROR IF ENABLED
U 348F. 8870.5C :13670 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 8
:13671
U 3490. 3640.95 :13672 LOOP.T1C.8: MOV LS[BIT0] TO WR[1]
U 3491. 3EFC.95 :13673 MOV WR[1] TO LS[SIZE] ;SIZE = WORD (01B)
U 3492. 9D0B.D5 :13674 MEM.REQ[WRITE.CSR] ADRS[T5.SUB] DT[SIZE] ;ISSUE MEM.REQ WITH SIZE DATA TYPE
U 3493. 36FC.15 :13675 MOV LS[MDT] TO WR[0]
U 3494. 364C.95 :13676 MOV LS[BIT6] TO WR[1] ;EXPECTED DATA = #00004000
U 3495. 0869.3C :13677 JSR [CHECK.RESULT] ;CHECK FOR BIT 6 SET
U 3496. 0B49.04 :13678 JMP [LOOP.T1C.8] ;LOOP ON ERROR IF ENABLED
U 3497. 8870.5C :13679 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 9
:13680
U 3498. 65FC.15 :13681 LOOP.T1C.9: CLR LS[SIZE] ;SIZE = BYTE (00B)
U 3499. 9D0B.D5 :13682 MEM.REQ[WRITE.CSR] ADRS[T5.SUB] DT[SIZE] ;ISSUE MEM.REQ WITH SIZE DATA TYPE
U 349A. 36FC.15 :13683 MOV LS[MDT] TO WR[0]
U 349B. 2F82.95 :13684 CLR WR[1] ;EXPECTED DATA = #00000000
U 349C. 0869.3C :13685 JSR [CHECK.RESULT] ;CHECK FOR BIT 6 & 7 CLEAR
U 349D. 8B49.84 :13686 JMP [LOOP.T1C.9] ;LOOP ON ERROR IF ENABLED
:13687 END.T1C:
    
```

Page "TEST 1D - Branch (skip) On Condition Codes Test (M8390 CPU Module)"

Test Description:

This test checks the previously untested skip conditions which come from the 8 to 1 mux feeding the SEQ.CTL PAL. ALU N, ALU V, ALU C, and PSL C are checked in this test. ALU Z has been tested previously and the other 3 inputs to the mux are tested later. The ALU and PSL codes are loaded from the Y bus for simplicity and the skip condition is tested for both skip if true and skip if false. The data path is as follows: WR 1 is loaded from LS with the value to put on the Y BUS for loading the ALU CC's. A MOV is executed to load this value. WR 2 is written from LS with the value for the PSL CC's (only PSL C is used however) and a load to the PSL CC's is executed, with the SCTL field set up to test the current skip function. Errors are reported when skip messes up. Data patterns vary, but the values of the codes not being tested are set up to the complement of the tested skip function.

Assumptions:

It is assumed that the previous condition code tests and the skip on ALU Z test have been successfully run. Also note that the expected and received data should be ignored in the error report as skip functions are being tested rather than data.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Set up ALU codes for N set, others clear. Clear PSL C and execute SCTL=N.CLR. Check for no skip.
- 3) Repeat with SCTL=N.SET. Check for skip.
- 4) Set up ALU codes for N clear, others set. Set PSL C and execute SCTL=N.CLR. Check for skip.
- 5) Repeat with SCTL=N.SET. Check for no skip.
- 6) Set up ALU codes for V set, others clear. Clear PSL C and execute SCTL=V.CLR. Check for no skip.
- 7) Repeat with SCTL=V.SET. Check for skip.
- 8) Set up ALU codes for V clear, others set. Set PSL C and execute SCTL=V.CLR. Check for skip.
- 9) Repeat with SCTL=V.SET. Check for no skip.
- 10) Set up ALU codes for C set, others clear. Clear PSL C and execute SCTL=C.SET. Check for skip.
- 11) Repeat with SCTL=C.CLR. Check for no skip.
- 12) Set up ALU codes for C clear, others set. Set PSL C and execute SCTL=C.CLR. Check for skip.
- 13) Repeat with SCTL=C.SET. Check for no skip.
- 14) Set up ALU codes for all bits clear. Set PSL C and execute SCTL=NOT.PSL.C. Check for no skip.
- 15) Set up ALU codes for all bits set. Clear PSL C and execute SCTL=NOT.PSL.C. Check for skip.

Errors:

:13743
:13744
:13745
:13746
:13747
:13748
:13749
:13750
:13751
:13752
:13753
:13754
:13755
:13756
:13757
:13758
:13759
:13760
:13761
:13762
:13763
:13764
:13765
:13766
:13767
:13768
:13769
:13770
:13771
:13772
:13773
:13774
:13775
:13776
:13777
:13778
:13779
:13780
:13781
:13782
:13783
:13784
:13785
:13786
:13787
:13788
:13789
:13790
:13791
:13792
:13793
:13794
:13795
:13796
:13797

Error 1 - SCTL=N.CLR skipped with N set.
Error 2 - SCTL=N.SET failed to skip with N set.
Error 3 - SCTL=N.CLR failed to skip with N clear.
Error 4 - SCTL=N.SET skipped with N clear.
Error 5 - SCTL=V.CLR skipped with V set.
Error 6 - SCTL=V.SET failed to skip with V set.
Error 7 - SCTL=V.CLR failed to skip with V clear.
Error 8 - SCTL=V.SET skipped with V clear.
Error 9 - SCTL=C.CLR skipped with C set.
Error A - SCTL=C.SET failed to skip with C set.
Error B - SCTL=C.CLR failed to skip with C clear.
Error C - SCTL=C.SET skipped with C clear.
Error D - SCTL=NOT.PSL.C skipped with PSL C set.
Error E - SCTL=NOT.PSL.C failed to skip with PSL C clear.

Debug:

Error 1 - First check the input to the 8 to 1 MUX feeding the SEQ.CTL PAL. ALU N H should be high and CSR 02, 01, and 00 should = 0 during the u-word that has SCTL=N.CLR. If the inputs are OK, check the output (pin 6) for a low. If this is incorrect, suspect the 8 to 1 mux, else check the inputs to the SEQ.CTL PAL. CSR 04 through CSR 00 should = 0. If this is correct, suspect the SEQ.CTL PAL.

Error 2 - Same as error 1, except CSR 04 through CSR 00 at the SEQ.CTL PAL should = 8(H) during the u-word that has SCTL=N.SET.

Error 3 - First check the input to the 8 to 1 MUX feeding the SEQ.CTL PAL. ALU N H should be low and CSR 02, 01, and 00 should = 0 during the u-word that has SCTL=N.CLR. If the inputs are OK, check the output (pin 6) for a high. If this is incorrect, suspect the 8 to 1 mux, else check the inputs to the SEQ.CTL PAL. CSR 04 through CSR 00 should = 0. If this is correct, suspect the SEQ.CTL PAL.

Error 4 - Same as error 3, except CSR 04 through CSR 00 at the SEQ.CTL PAL should = 8(H) during the u-word that has SCTL=N.SET.

Error 5 - First check the input to the 8 to 1 MUX feeding the SEQ.CTL PAL. ALU V H should be high and CSR 02, 01, and 00 should = 2 during the u-word that has SCTL=V.CLR. If the inputs are OK, check the output (pin 6) for a low. If this is incorrect, suspect the 8 to 1 mux, else check the inputs to the SEQ.CTL PAL. CSR 04 through CSR 00 should = 2. If this is correct, suspect the SEQ.CTL PAL.

Error 6 - Same as error 5, except CSR 04 through CSR 00 at the SEQ.CTL PAL should = A(H) during the u-word that has SCTL=V.SET.

Error 7 - First check the input to the 8 to 1 MUX feeding the SEQ.CTL PAL. ALU V H should be low and CSR 02, 01, and 00 should = 2 during the u-word that has SCTL=V.CLR. If the inputs are OK, check the output (pin 6) for a high. If this is incorrect, suspect the 8 to 1 mux, else check the inputs to the SEQ.CTL PAL. CSR 04 through CSR 00 should = 2. If this is correct, suspect the SEQ.CTL PAL.

```

:13798      Error 8 - Same as error 7, except CSR 04 through CSR 00 at the SEQ.CTL
:13799      PAL should = A(H) during the u-word that has SCTL=V.SET.
:13800
:13801      Error 9 - First check the input to the 8 to 1 MUX feeding the SEQ.CTL
:13802      PAL. ALU C L should be low and CSR 02, 01, and 00 should = 3 during
:13803      the u-word that has SCTL=C.SET. If the inputs are OK, check the
:13804      output (pin 6) for a high. If this is incorrect, suspect the 8 to 1
:13805      mux, else check the inputs to the SEQ.CTL PAL. CSR 04 through CSR 00
:13806      should = 3. If this is correct, suspect the SEQ.CTL PAL.
:13807
:13808      Error A - Same as error 9, except CSR 04 through CSR 00 at the
:13809      SEQ.CTL PAL should = B(H) during the u-word that has SCTL=C.CLR.
:13810
:13811      Error B - First check the input to the 8 to 1 MUX feeding the SEQ.CTL
:13812      PAL. ALU C L should be high and CSR 02, 01, and 00 should = 3(H)
:13813      during the u-word that has SCTL=C.CLR. If the inputs are OK, check
:13814      the output (pin 6) for a low. If this is incorrect, suspect the 8 to
:13815      1 mux, else check the inputs to the SEQ.CTL PAL. CSR 04 through CSR
:13816      00 should = B (H). If this is correct, suspect the SEQ.CTL PAL.
:13817
:13818      Error C - Same as error 12, except CSR 04 through CSR 00 at the
:13819      SEQ.CTL PAL should = 3(H) during the u-word that has SCTL=C.SET.
:13820
:13821      Error D - First check the input to the 8 to 1 MUX feeding the SEQ.CTL
:13822      PAL. PSL C H should be high and CSR 02, 01, and 00 should = 0 during
:13823      the u-word that has SCTL=N.CLR. If the inputs are OK, check the
:13824      output (pin 6) for a low. If this is incorrect, suspect the 8 to 1
:13825      mux, else check the inputs to the SEQ.CTL PAL. CSR 04 through CSR 00
:13826      should = 0. If this is correct, suspect the SEQ.CTL PAL.
:13827
:13828      Error E - First check the input to the 8 to 1 MUX feeding the SEQ.CTL
:13829      PAL. PSL C H should be low and CSR 02, 01, and 00 should = 0 during
:13830      the u-word that has SCTL=N.CLR. If the inputs are OK, check the
:13831      output (pin 6) for a high. If this is incorrect, suspect the 8 to 1
:13832      mux, else check the inputs to the SEQ.CTL PAL. CSR 04 through CSR 00
:13833      should = 0. If this is correct, suspect the SEQ.CTL PAL.
:13834
:13835
:13836

```

```

U 349E, B65E,15
U 349F, 3E80,15
U 34A0, 10E0,15
U 34A1, 8B4A,14
U 34A2, E58A,15
U 34A3, 65A0,15
U 34A4, 2F80,15
U 34A5, 2040,15
U 34A6, BE82,15
U 34A7, A3C0,15
U 34A8, 3E8C,15
U 34A9, B66E,15
U 34AA, 3E80,15

```

T.1D:

WAIT.T1D.0:

LOOP.T1D.1:

```

MOV LS[BEGIN.TEST] TO WR[0]      ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN]               ;BIT15 INDICATES START OF TEST TO CONSOLE
                                ;ISSUE CPU ATTN TO CONSOLE
JMP [WAIT.T1D.0]                 ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR LS[ERROR.MASK]               ;CLEAR ERROR MASK.
CLR LS[PREVIOUS.ERROR]           ;CLEAR PREVIOUS ERROR NUMBER
CLR WR[0]
INC WR[0]                        ;SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER]    ;SET ERROR NUMBER TO FIRST ERROR
ROL WR[0]                        ;SET WRO TO 2
MOV WR[0] TO LS[MODULE.NUM]      ;SET UP MODULE CODE FOR CPU MODULE
MOV LS[BIT23] TO WR[0]
MOV WR[0] TO LS[CONTROL.STATUS] ;BIT23 = EXPECTED AND RECEIVED DATA N/A

```


U 34AB, 3646,95	:13853	MOV LS[BIT3] TO WR[1]	
U 34AC, BFFC,95	:13854	MOV WR[1] TO LS[ALU.CC]	:SET N BIT IN ALU CODE
U 34AD, E5FE,15	:13855	CLR LS[PSL.CC]	:INSURE THAT PSL C IS CLEAR
U 34AE, 5B00,00	:13856	SKIP,IF[N,CLR]	
U 34AF, 8B4B,34	:13857	JMP [T1D,2]	:SKIP OVER ERROR CALL
U 34B0, 2F80,15	:13858	CLR WR[0]	:ERROR IF HERE
U 34B1, 0869,3C	:13859	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13860		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 34B2, 8B4A,B4	:13861	JMP [LOOP.T1D.1]	:LOOP ON ERROR IF ENABLE
	:13862		
U 34B3, 8870,5C	:13863	T1D.2: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 2
	:13864	LOOP.T1D.2:	
U 34B4, 3646,95	:13865	MOV LS[BIT3] TO WR[1]	
U 34B5, BFFC,95	:13866	MOV WR[1] TO LS[ALU.CC]	:SET N BIT IN ALU CODE
U 34B6, E5FE,15	:13867	CLR LS[PSL.CC]	:INSURE THAT PSL C IS CLEAR
U 34B7, DB00,08	:13868	SKIP,IF[N,SET]	
U 34B8, 5B00,1E	:13869	SKIP	:ERROR -- SKIP NEXT INSTRUCTION
U 34B9, 0B4B,D4	:13870	JMP [T1D,3]	:SKIP OVER ERROR CALL
U 34BA, 2F80,15	:13871	CLR WR[0]	:ERROR IF HERE
U 34BB, 0869,3C	:13872	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13873		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 34BC, 0B4B,44	:13874	JMP [LOOP.T1D.2]	:LOOP ON ERROR IF ENABLE
	:13875		
U 34BD, 8870,5C	:13876	T1D.3: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 3
	:13877	LOOP.T1D.3:	
U 34BE, 5F46,95	:13878	MCOM LS[BIT3] TO WR[1]	
U 34BF, BFFC,95	:13879	MOV WR[1] TO LS[ALU.CC]	:SET ALL BUT N IN ALU CODES
U 34C0, 3640,95	:13880	MOV LS[BIT0] TO WR[1]	
U 34C1, BEFE,95	:13881	MOV WR[1] TO LS[PSL.CC]	:SET PSL C
U 34C2, 5B00,00	:13882	SKIP,IF[N,CLR]	
U 34C3, 5B00,1E	:13883	SKIP	:ERROR -- SKIP NEXT INSTRUCTION
U 34C4, 8B4C,84	:13884	JMP [T1D,4]	:SKIP OVER ERROR CALL
U 34C5, 2F80,15	:13885	CLR WR[0]	:ERROR IF HERE
U 34C6, 0869,3C	:13886	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13887		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 34C7, 0B4B,E4	:13888	JMP [LOOP.T1D.3]	:LOOP ON ERROR IF ENABLE
	:13889		
U 34C8, 8870,5C	:13890	T1D.4: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 4
	:13891	LOOP.T1D.4:	
U 34C9, 5F46,95	:13892	MCOM LS[BIT3] TO WR[1]	
U 34CA, BFFC,95	:13893	MOV WR[1] TO LS[ALU.CC]	:SET ALL BUT N IN ALU CODES
U 34CB, 3640,95	:13894	MOV LS[BIT0] TO WR[1]	
U 34CC, BEFE,95	:13895	MOV WR[1] TO LS[PSL.CC]	:SET PSL C
U 34CD, DB00,08	:13896	SKIP,IF[N,SET]	
U 34CE, 0B4D,24	:13897	JMP [T1D,5]	:SKIP OVER ERROR CALL
U 34CF, 2F80,15	:13898	CLR WR[0]	:ERROR IF HERE
U 34D0, 0869,3C	:13899	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13900		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 34D1, 0B4C,94	:13901	JMP [LOOP.T1D.4]	:LOOP ON ERROR IF ENABLE
	:13902		
U 34D2, 8870,5C	:13903	T1D.5: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 5
	:13904	LOOP.T1D.5:	
U 34D3, B642,95	:13905	MOV LS[BIT1] TO WR[1]	
U 34D4, BFFC,95	:13906	MOV WR[1] TO LS[ALU.CC]	:SET V BIT IN ALU CODE
U 34D5, E5FE,15	:13907	CLR LS[PSL.CC]	:INSURE THAT PSL C IS CLEAR

U 34D6, DB00.02	:13908	SKIP,IF[V.CLR]	
U 34D7, 0B4D.B4	:13909	JMP [T1D.6]	:SKIP OVER ERROR CALL
U 34D8, 2F80.15	:13910	CLR WR[0]	:ERROR IF HERE
U 34D9, 0869.3C	:13911	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13912		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 34DA, 8B4D.34	:13913	JMP [LOOP.T1D.5]	:LOOP ON ERROR IF ENABLE
	:13914		
U 34DB, 8870.5C	:13915	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 6
	:13916		
U 34DC, B642.95	:13917	MOV LS[BIT1] TO WR[1]	
U 34DD, BFFC.95	:13918	MOV WR[1] TO LS[ALU.CC]	:SET V BIT IN ALU CODE
U 34DE, E5FE.15	:13919	CLR LS[PSL.CC]	:INSURE THAT PSL C IS CLEAR
U 34DF, 5B00.0A	:13920	SKIP,IF[V.SET]	
U 34E0, 5B00.1E	:13921	SKIP	:ERROR -- SKIP NEXT INSTRUCTION
U 34E1, 8B4E.54	:13922	JMP [T1D.7]	:SKIP OVER ERROR CALL
U 34E2, 2F80.15	:13923	CLR WR[0]	:ERROR IF HERE
U 34E3, 0869.3C	:13924	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13925		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 34E4, 8B4D.C4	:13926	JMP [LOOP.T1D.6]	:LOOP ON ERROR IF ENABLE
	:13927		
U 34E5, 8870.5C	:13928	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 7
	:13929		
U 34E6, DF42.95	:13930	MCOM LS[BIT1] TO WR[1]	
U 34E7, BFFC.95	:13931	MOV WR[1] TO LS[ALU.CC]	:SET ALL BUT V IN ALU CODES
U 34E8, 3640.95	:13932	MOV LS[BIT0] TO WR[1]	
U 34E9, BEFE.95	:13933	MOV WR[1] TO LS[PSL.CC]	:SET PSL C
U 34EA, DB00.02	:13934	SKIP,IF[V.CLR]	
U 34EB, 5B00.1E	:13935	SKIP	:ERROR -- SKIP NEXT INSTRUCTION
U 34EC, 0B4F.04	:13936	JMP [T1D.8]	:SKIP OVER ERROR CALL
U 34ED, 2F80.15	:13937	CLR WR[0]	:ERROR IF HERE
U 34EE, 0869.3C	:13938	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13939		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 34EF, 8B4E.64	:13940	JMP [LOOP.T1D.7]	:LOOP ON ERROR IF ENABLE
	:13941		
U 34F0, 8870.5C	:13942	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 8
	:13943		
U 34F1, DF42.95	:13944	MCOM LS[BIT1] TO WR[1]	
U 34F2, BFFC.95	:13945	MOV WR[1] TO LS[ALU.CC]	:SET ALL BUT V IN ALU CODES
U 34F3, 3640.95	:13946	MOV LS[BIT0] TO WR[1]	
U 34F4, BEFE.95	:13947	MOV WR[1] TO LS[PSL.CC]	:SET PSL C
U 34F5, 5B00.0A	:13948	SKIP,IF[V.SET]	
U 34F6, 0B4F.A4	:13949	JMP [T1D.9]	:SKIP OVER ERROR CALL
U 34F7, 2F80.15	:13950	CLR WR[0]	:ERROR IF HERE
U 34F8, 0869.3C	:13951	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13952		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 34F9, 8B4F.14	:13953	JMP [LOOP.T1D.8]	:LOOP ON ERROR IF ENABLE
	:13954		
U 34FA, 8870.5C	:13955	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 9
	:13956		
U 34FB, 3640.95	:13957	MOV LS[BIT0] TO WR[1]	
U 34FC, BFFC.95	:13958	MOV WR[1] TO LS[ALU.CC]	:SET C BIT IN ALU CODE
U 34FD, E5FE.15	:13959	CLR LS[PSL.CC]	:INSURE THAT PSL C IS CLEAR
U 34FE, DB00.0B	:13960	SKIP,IF[C.CLR]	
U 34FF, 8B50.34	:13961	JMP [T1D.A]	:SKIP OVER ERROR CALL
U 3500, 2F80.15	:13962	CLR WR[0]	:ERROR IF HERE

U 3501, 0869,3C	:13963	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13964		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 3502, 8B4F,B4	:13965	JMP [LOOP.T1D.9]	:LOOP ON ERROR IF ENABLE
	:13966		
U 3503, 8870,5C	:13967	T1D.A: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO A
	:13968	LOOP.T1D.A:	
U 3504, 3640,95	:13969	MOV LS[BIT0] TO WR[1]	
U 3505, BFFC,95	:13970	MOV WR[1] TO LS[ALU.CC]	:SET C BIT IN ALU CODE
U 3506, EFCE,15	:13971	CLR LS[PSL.CC]	:INSURE THAT PSL C IS CLEAR
U 3507, 5B00,03	:13972	SKIP.IF[C.SET]	
U 3508, 5B00,1E	:13973	SKIP	:ERROR -- SKIP NEXT INSTRUCTION
U 3509, 0B50,D4	:13974	JMP [T1D.B]	:SKIP OVER ERROR CALL
U 350A, 2F80,15	:13975	CLR WR[0]	:ERROR IF HERE
U 350B, 0869,3C	:13976	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13977		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 350C, 0B50,44	:13978	JMP [LOOP.T1D.A]	:LOOP ON ERROR IF ENABLE
	:13979		
U 350D, 8870,5C	:13980	T1D.B: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO B
	:13981	LOOP.T1D.B:	
U 350E, 5F40,95	:13982	MCOM LS[BIT0] TO WR[1]	
U 350F, BFFC,95	:13983	MOV WR[1] TO LS[ALU.CC]	:SET ALL BUT C IN ALU CODES
U 3510, 3640,95	:13984	MOV LS[BIT0] TO WR[1]	
U 3511, BEFE,95	:13985	MOV WR[1] TO LS[PSL.CC]	:SET PSL C
U 3512, DB00,0B	:13986	SKIP.IF[C.CLR]	
U 3513, 5B00,1E	:13987	SKIP	:ERROR -- SKIP NEXT INSTRUCTION
U 3514, 8B51,84	:13988	JMP [T1D.C]	:SKIP OVER ERROR CALL
U 3515, 2F80,15	:13989	CLR WR[0]	:ERROR IF HERE
U 3516, 0869,3C	:13990	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:13991		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 3517, 0B50,E4	:13992	JMP [LOOP.T1D.B]	:LOOP ON ERROR IF ENABLE
	:13993		
U 3518, 8870,5C	:13994	T1D.C: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO C
	:13995	LOOP.T1D.C:	
U 3519, 5F40,95	:13996	MCOM LS[BIT0] TO WR[1]	
U 351A, BFFC,95	:13997	MOV WR[1] TO LS[ALU.CC]	:SET ALL BUT C IN ALU CODES
U 351B, 3640,95	:13998	MOV LS[BIT0] TO WR[1]	
U 351C, BEFE,95	:13999	MOV WR[1] TO LS[PSL.CC]	:SET PSL C
U 351D, 5B00,03	:14000	SKIP.IF[C.SET]	
U 351E, 8B52,24	:14001	JMP [T1D.D]	:SKIP OVER ERROR CALL
U 351F, 2F80,15	:14002	CLR WR[0]	:ERROR IF HERE
U 3520, 0869,3C	:14003	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14004		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 3521, 0B51,94	:14005	JMP [LOOP.T1D.C]	:LOOP ON ERROR IF ENABLE
	:14006		
U 3522, 8870,5C	:14007	T1D.D: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO D
	:14008	LOOP.T1D.D:	
U 3523, 2F82,95	:14009	CLR WR[1]	
U 3524, BFFC,95	:14010	MOV WR[1] TO LS[ALU.CC]	:CLEAR ALL ALU CODES
U 3525, 3640,95	:14011	MOV LS[BIT0] TO WR[1]	
U 3526, BEFE,95	:14012	MOV WR[1] TO LS[PSL.CC]	:SET PSL C
U 3527, DB00,04	:14013	SKIP.IF[NOT.PSL.C]	
U 3528, 0B52,C4	:14014	JMP [T1D.E]	:SKIP OVER ERROR CALL
U 3529, 2F80,15	:14015	CLR WR[0]	:ERROR IF HERE
U 352A, 0869,3C	:14016	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14017		:ERRONEOUS DATA TO FLAG SKIP ERROR

G 8
3-MAR-82 09:34:33 FICHE 2 Frame G8 Sequence 303 Page 297
ENKCA Micro-diagnostic for DAP module Rev 01.00

```

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC
MICRO2 1L(03) TEST 1D - Branch (skip) On Condition Codes Test (M8390 CPU Module)

U 352B, 0B52,34 :14018      JMP [LOOP.T1D.D]          ;LOOP ON ERROR IF ENABLE
:14019
U 352C, 8870,5C :14020      T1D.E: JSR [INC.EN]          ;INCREMENT ERROR NUMBER TO E
:14021      LOOP.T1D.E:
U 352D, 369E,95 :14022      MOV LS[ONES] TO WR[1]
U 352E, BFFC,95 :14023      MOV WR[1] TO LS[ALU.CC]      ;SET ALL CODES IN ALU CODES
U 352F, ESFE,15 :14024      CLR LS[PSL.CC]          ;INSURE THAT PSL C IS CLEAR
U 3530, DB00,04 :14025      SKIP.IF[NOT.PSL.C]
U 3531, 5B00,1E :14026      SKIP
U 3532, 8B53,64 :14027      JMP [END.T1D]          ;ERROR -- SKIP NEXT INSTRUCTION
U 3533, 2F80,15 :14028      CLR WR[0]          ;SKIP OVER ERROR CALL
U 3534, 0B69,3C :14029      JSR [CHECK.RESULT]      ;ERROR IF HERE
:14030      ;CALL ERROR ROUTINE WITH DELIBERATELY
:14031      ;ERRONEOUS DATA TO FLAG SKIP ERROR
:14032      ;LOOP ON ERROR IF ENABLE
      END.T1D: JMP [LOOP.T1D.E]

```


:14033
:14034
:14035
:14036
:14037
:14038
:14039
:14040
:14041
:14042
:14043
:14044
:14045
:14046
:14047
:14048
:14049
:14050
:14051
:14052
:14053
:14054
:14055
:14056
:14057
:14058
:14059
:14060
:14061
:14062
:14063
:14064
:14065
:14066
:14067
:14068
:14069
:14070
:14071
:14072
:14073
:14074
:14075
:14076
:14077
:14078
:14079
:14080
:14081
:14082
:14083
:14084
:14085
:14086
:14087

Page "TEST 1E - Conditional Jump Tests (M8390 CPU Module)"

++

Test Description:

This test checks the previously untested jump conditions which come from the 8 to 1 mux feeding the SEQ.CTL PAL. ALU N, ALU V, and ALU C are used in this test. The mux itself and its inputs have been tested in the previous test. This test checks the SEQ.CTL PAL for the jump conditions. The ALU codes are loaded from the Y bus for simplicity and the jump condition is tested for both jump if true and jump if false. The data path is as follows: WR 1 is loaded from LJ with the value to put on the Y BUS for loading the ALU CC's. A MOV is executed to load this value, with the JCTL field set up to test the current jump function. Errors are reported when jump messes up. Data patterns vary, but the values of the codes not being tested are set up to the complement of the tested skip function.

Assumptions:

It is assumed that the previous conditional skip test has run successfully. Note that the expected and received data printed in the error report should be ignored, as this test is checking for jumps, not data.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Set up ALU codes for N set, others clear. Execute JCTL=N.CLR. Check for no jump.
- 3) Repeat with JCTL=N.SET. Check for jump.
- 4) Set up ALU codes for N clear, others set. Execute JCTL=N.CLR. Check for jump.
- 5) Repeat with JCTL=N.SET. Check for no jump.
- 6) Set up ALU codes for V set, others clear. Execute JCTL=V.CLR. Check for no jump.
- 7) Repeat with JCTL=V.SET. Check for jump.
- 8) Set up ALU codes for V clear, others set. Execute JCTL=V.CLR. Check for jump.
- 9) Repeat with JCTL=V.SET. Check for no jump.
- 10) Set up ALU codes for C set, others clear. Execute JCTL=C.SET. Check for jump.
- 11) Repeat with JCTL=C.CLR. Check for no jump.
- 12) Set up ALU codes for C clear, others set. Execute JCTL=C.CLR. Check for jump.
- 13) Repeat with JCTL=C.SET. Check for no jump.

Errors:

- Error 1 - JCTL=N.CLR jumped with N set.
Error 2 - JCTL=N.SET failed to jump with N set.
Error 3 - JCTL=N.CLR failed to jump with N clear.
Error 4 - JCTL=N.SET jumped with N clear.
Error 5 - JCTL=V.CLR jumped with V set.
Error 6 - JCTL=V.SET failed to jump with V set.

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 1E - Condition 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 1E - Conditional Jump Tests (M8390 CPU Module)

Fiche 2 Frame 18

Sequence 305
Rev 01.00 Page 299

:14088 : Error 7 - JCTL=V.CLR failed to jump with V clear.
:14089 : Error 8 - JCTL=V.SET jumped with V clear.
:14090 : Error 9 - JCTL=C.SET failed to jump with C set.
:14091 : Error A - JCTL=C.CLR jumped with C set.
:14092 : Error B - JCTL=C.CLR failed to jump with C clear.
:14093 : Error C - JCTL=C.SET jumped with C clear.
:14094 :
:14095 :
:14096 :
:14097 :
:14098 :
:14099 :
:14100 :
:14101 :
:14102 :
:14103 :
:14104 :
:14105 :
:14106 :
:14107 :
:14108 :
:14109 :
:14110 :
:14111 :
:14112 :
:14113 :
:14114 :
:14115 :
:14116 :
:14117 :
:14118 :
:14119 :
:14120 :
:14121 :
:14122 :
:14123 :
:14124 :
:14125 :
:14126 :
:14127 :
:14128 :
:14129 :
:14130 :
:14131 :
:14132 :
:14133 :
:14134 :
:14135 :
:14136 :
:14137 :
:14138 :
:14139 :
:14140 :
:14141 :
:14142 :

Debug:

Errors 1 through C - Any of these errors most likely indicates a problem with the SEQ.CTL PAL. All the inputs to this PAL were used in the previous conditional skip test and the jump logic itself has been previously tested. Suspect a bad PAL if a failure occurs.

T.1E:

U 3536, B65E,15 :14104 MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
U 3537, 3E80,15 :14105 MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
:14106 : ;BIT15 INDICATES START OF TEST TO CONSOLE
U 3538, 10E0,15 :14107 MISC [SET.CP.ATTN] ;ISSUE CPU ATTN TO CONSOLE
:14108 :
WAIT.T1E.0: :14109 JMP [WAIT.T1E.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 3539, 8B53,94 :14109 CLR LS[ERROR.MASK] ;CLEAR ERROR MASK.
U 353A, E58A,15 :14110 CLR LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
U 353B, 65A0,15 :14111 CLR WR[0]
U 353C, 2F80,15 :14112 INC WR[0] ;SET WRO TO 1
U 353D, 2040,15 :14113 MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
U 353E, BE82,15 :14114 ROL WR[0] ;SET WRO TO 2
U 353F, A3C0,15 :14115 MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
U 3540, 3E8C,15 :14116 MOV LS[BIT23] TO WR[0]
U 3541, B66E,15 :14117 MOV WR[0] TO LS[CONTROL.STATUS] ;BIT23 = EXPECTED AND RECEIVED DATA N/A
U 3542, 3E80,15 :14118 :
:14119 :
LOOP.T1E.1: :14120 MOV LS[BIT3] TO WR[1]
U 3543, 3646,95 :14120 MOV WR[1] TO LS[ALU.CC] ;SET N BIT IN ALU CODE
U 3544, BFFC,95 :14121 JMP IF[N.CLR] TO [T1E.ERROR.1]
U 3545, 0B54,70 :14122 JMP [T1E.2] ;JUMP OVER ERROR CALL
U 3546, 0B54,A4 :14123 :
:14124 :
T1E.ERROR.1: :14125 CLR WR[0]
U 3547, 2F80,15 :14125 JSR [CHECK.RESULT] ;CALL ERROR ROUTINE WITH DELIBERATELY
U 3548, 0B69,3C :14126 : ;ERRONEOUS DATA TO FLAG JUMP ERROR
:14127 : ;LOOP ON ERROR IF ENABLED
U 3549, 0B54,34 :14128 JMP [LOOP.T1E.1]
:14129 :
T1E.2: :14129 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
U 354A, 8870,5C :14130 :
:14131 :
LOOP.T1E.2: :14132 MOV LS[BIT3] TO WR[1]
U 354B, 3646,95 :14132 MOV WR[1] TO LS[ALU.CC] ;SET N BIT IN ALU CODE
U 354C, BFFC,95 :14133 JMP IF[N.SET] TO [T1E.3]
U 354D, 0B55,18 :14134 CLR WR[0]
U 354E, 2F80,15 :14135 JSR [CHECK.RESULT] ;ERROR IF HERE
U 354F, 0B69,3C :14136 : ;CALL ERROR ROUTINE WITH DELIBERATELY
:14137 : ;ERRONEOUS DATA TO FLAG JUMP ERROR
:14138 : ;LOOP ON ERROR IF ENABLED
U 3550, 8B54,B4 :14138 JMP [LOOP.T1E.2]
:14139 :
T1E.3: :14139 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 3
U 3551, 8870,5C :14140 :
:14141 :
LOOP.T1E.3: :14141 MCOM LS[BIT3] TO WR[1]
U 3552, 5F46,95 :14142

U 3553, BFFC.95	:14143	MOV WR[1] TO LS[ALU.CC]	:SET ALL BUT N BIT IN ALU CODES
U 3554, 8B55.80	:14144	JMP.IF[N.CLR] TO [T1E.4]	
U 3555, 2F80.15	:14145	CLR WR[0]	:ERROR IF HERE
U 3556, 0869.3C	:14146	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14147		:ERRONEOUS DATA TO FLAG JUMP ERROR
U 3557, 0B55.24	:14148	JMP [LOOP.T1E.3]	:LOOP ON ERROR IF ENABLED
	:14149		
U 3558, 8870.5C	:14150	T1E.4: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 4
	:14151	LOOP.T1E.4:	
U 3559, 5F46.95	:14152	MCOM LS[BIT3] TO WR[1]	
U 355A, BFFC.95	:14153	MOV WR[1] TO LS[ALU.CC]	:SET ALL BUT N BIT IN ALU CODES
U 355B, 0B55.F0	:14154	JMP.IF[N.CLR] TO [T1E.5]	:SHOULD JUMP OVER ERROR
U 355C, 2F80.15	:14155	CLR WR[0]	
U 355D, 0869.3C	:14156	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14157		:ERRONEOUS DATA TO FLAG JUMP ERROR
U 355E, 8B55.94	:14158	JMP [LOOP.T1E.4]	:LOOP ON ERROR IF ENABLED
	:14159		
U 355F, 8870.5C	:14160	T1E.5: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 5
	:14161	LOOP.T1E.5:	
U 3560, B642.95	:14162	MOV LS[BIT1] TO WR[1]	
U 3561, BFFC.95	:14163	MOV WR[1] TO LS[ALU.CC]	:SET V BIT IN ALU CODE
U 3562, 0B56.42	:14164	JMP.IF[V.CLR] TO [T1E.ERROR.5]	
U 3563, 0B56.74	:14165	JMP [T1E.6]	:JUMP OVER ERROR CALL
	:14166	T1E.ERROR.5:	
U 3564, 2F80.15	:14167	CLR WR[0]	
U 3565, 0869.3C	:14168	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14169		:ERRONEOUS DATA TO FLAG JUMP ERROR
U 3566, 8B56.04	:14170	JMP [LOOP.T1E.5]	:LOOP ON ERROR IF ENABLED
	:14171		
U 3567, 8870.5C	:14172	T1E.6: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 6
	:14173	LOOP.T1E.6:	
U 3568, B642.95	:14174	MOV LS[BIT1] TO WR[1]	
U 3569, BFFC.95	:14175	MOV WR[1] TO LS[ALU.CC]	:SET V BIT IN ALU CODE
U 356A, 8B56.EA	:14176	JMP.IF[V.SET] TO [T1E.7]	
U 356B, 2F80.15	:14177	CLR WR[0]	:ERROR IF HERE
U 356C, 0869.3C	:14178	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14179		:ERRONEOUS DATA TO FLAG JUMP ERROR
U 356D, 0B56.84	:14180	JMP [LOOP.T1E.6]	:LOOP ON ERROR IF ENABLED
	:14181		
U 356E, 8870.5C	:14182	T1E.7: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 7
	:14183	LOOP.T1E.7:	
U 356F, DF42.95	:14184	MCOM LS[BIT1] TO WR[1]	
U 3570, BFFC.95	:14185	MOV WR[1] TO LS[ALU.CC]	:SET ALL BUT V BIT IN ALU CODES
U 3571, 0B57.52	:14186	JMP.IF[V.CLR] TO [T1E.8]	
U 3572, 2F80.15	:14187	CLR WR[0]	:ERROR IF HERE
U 3573, 0869.3C	:14188	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14189		:ERRONEOUS DATA TO FLAG JUMP ERROR
U 3574, 8B56.F4	:14190	JMP [LOOP.T1E.7]	:LOOP ON ERROR IF ENABLED
	:14191		
U 3575, 8870.5C	:14192	T1E.8: JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 8
	:14193	LOOP.T1E.8:	
U 3576, DF42.95	:14194	MCOM LS[BIT1] TO WR[1]	
U 3577, BFFC.95	:14195	MOV WR[1] TO LS[ALU.CC]	:SET ALL BUT V BIT IN ALU CODES
U 3578, 0B57.C2	:14196	JMP.IF[V.CLR] TO [T1E.9]	:SHOULD JUMP OVER ERROR
U 3579, 2F80.15	:14197	CLR WR[0]	

```

U 357A, 0869,3C :14198      JSR [CHECK.RESULT]      ;CALL ERROR ROUTINE WITH DELIBERATELY
:14199                      ;ERRONEOUS DATA TO FLAG JUMP ERROR
U 357B, 0857,64 :14200      JMP [LOOP.T1E.8]      ;LOOP ON ERROR IF ENABLED
:14201
U 357C, 8870,5C :14202      T1E.9: JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO 9
:14203      LOOP.T1E.9:
U 357D, 3640,95 :14204          MOV LS[BIT0] TO WR[1]
U 357E, BFFC,95 :14205          MOV WR[1] TO LS[ALU.CC]      ;SET C BIT IN ALU CODE
U 357F, 8B58,1B :14206          JMP.[IF[C.CLR] TO [T1E.ERROR.9]
U 3580, 8B58,44 :14207          JMP [T1E.A]      ;JUMP OVER ERROR CALL
:14208      T1E.ERROR.9:
U 3581, 2F80,15 :14209          CLR WR[0]
U 3582, 0869,3C :14210          JSR [CHECK.RESULT]      ;CALL ERROR ROUTINE WITH DELIBERATELY
:14211                      ;ERRONEOUS DATA TO FLAG JUMP ERROR
U 3583, 8B57,D4 :14212          JMP [LOOP.T1E.9]      ;LOOP ON ERROR IF ENABLED
:14213
U 3584, 8870,5C :14214      T1E.A: JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO A
:14215      LOOP.T1E.A:
U 3585, 3640,95 :14216          MOV LS[BIT0] TO WR[1]
U 3586, BFFC,95 :14217          MOV WR[1] TO LS[ALU.CC]      ;SET C BIT IN ALU CODE
U 3587, 0B58,B3 :14218          JMP.[IF[C.SET] TO [T1E.B]
U 3588, 2F80,15 :14219          CLR WR[0]
U 3589, 0869,3C :14220          JSR [CHECK.RESULT]      ;ERROR IF HERE
:14221                      ;CALL ERROR ROUTINE WITH DELIBERATELY
:14222                      ;ERRONEOUS DATA TO FLAG JUMP ERROR
U 358A, 0B58,54 :14223          JMP [LOOP.T1E.A]      ;LOOP ON ERROR IF ENABLED
:14224
U 358B, 8870,5C :14225      T1E.B: JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO B
:14226      LOOP.T1E.B:
U 358C, 5F40,95 :14227          MCOM LS[BIT0] TO WR[1]
U 358D, BFFC,95 :14228          MOV WR[1] TO LS[ALU.CC]      ;SET ALL BUT C BIT IN ALU CODES
U 358E, 0B59,2B :14229          JMP.[IF[C.CLR] TO [T1E.C]
U 358F, 2F80,15 :14230          CLR WR[0]
U 3590, 0869,3C :14231          JSR [CHECK.RESULT]      ;ERROR IF HERE
:14232                      ;CALL ERROR ROUTINE WITH DELIBERATELY
:14233                      ;ERRONEOUS DATA TO FLAG JUMP ERROR
U 3591, 0B58,C4 :14234          JMP [LOOP.T1E.B]      ;LOOP ON ERROR IF ENABLED
:14235
U 3592, 8870,5C :14236      T1E.C: JSR [INC.EN]      ;INCREMENT ERROR NUMBER TO C
:14237      LOOP.T1E.C:
U 3593, 5F40,95 :14238          MCOM LS[BIT0] TO WR[1]
U 3594, BFFC,95 :14239          MOV WR[1] TO LS[ALU.CC]      ;SET ALL BUT C BIT IN ALU CODES
U 3595, 8B59,9B :14240          JMP.[IF[C.CLR] TO [END.T1E]      ;SHOULD JUMP OVER ERROR CALL
U 3596, 2F80,15 :14241          CLR WR[0]
U 3597, 0869,3C :14242          JSR [CHECK.RESULT]      ;CALL ERROR ROUTINE WITH DELIBERATELY
:14243                      ;ERRONEOUS DATA TO FLAG JUMP ERROR
U 3598, 8B59,34 :14244          JMP [LOOP.T1E.C]      ;LOOP ON ERROR IF ENABLED
:14245      END.T1E:
  
```


Page "TEST 1F - Skip Logic Using MUX & INCR CTL PAL Test (M8390 CPU Module)"

Test Description:

This test checks the skip conditions which originate at the MUX & INCR CTL PAL. The skip conditions tested are those using CONSOLE ACK, CONSOLE ATTN, and the XOR of ALU V and ALU N (signed less than). The STATE REG inputs will be checked in the next test. The two CONSOLE signals will be set or cleared by issuing CPU ATTN to the console monitor with bit 16 of the control and status word set. Bits 20 and 21 of this word will indicate the state to set CONSOLE ATTN and CONSOLE ACK respectively. ALU V and ALU N will be set up using the Y BUS. The data path is simply through the MUX & INCR CTL PAL to the three input negated 'OR' gate which outputs INCR CIN H.

Assumptions:

It is assumed that the skip logic and 2901 tests have run successfully prior to this test.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Clear ALU N and ALU V and execute SCTL=LSS. Check for no skip.
- 3) Set ALU N and ALU V and execute SCTL=LSS. Check for no skip.
- 4) Set ALU N, clear ALU V and execute SCTL=LSS. Check for skip.
- 5) Set ALU V, clear ALU N and execute SCTL=LSS. Check for skip.
- 6) Set CONSOLE ACK and clear CONSOLE ATTN by issuing CPU ATTN with bits 16 and 21 set and bit 20 clear in the control and status word.
- 7) Execute SCTL=CONSOLE.ACK and check for skip.
- 8) Execute SCTL=CONSOLE.ATTN and check for no skip.
- 9) Clear CONSOLE ACK and set CONSOLE ATTN by issuing CPU ATTN with bits 16 and 20 set and bit 21 clear in the control and status word.
- 10) Execute SCTL=CONSOLE.ACK and check for no skip.
- 11) Execute SCTL=CONSOLE.ATTN and check for skip.
- 12) Call the console to clear CONSOLE ACK and CONSOLE ATTN (for house-keeping purposes).

Errors:

- Error 1 - SCTL=LSS (ALU N XOR ALU V) skipped with ALU N and ALU V both set.
- Error 2 - SCTL=LSS (ALU N XOR ALU V) skipped with ALU N and ALU V both clear.
- Error 3 - SCTL=LSS (ALU N XOR ALU V) failed to skip with ALU N set and ALU V clear.
- Error 4 - SCTL=LSS (ALU N XOR ALU V) failed to skip with ALU V set and ALU N clear.
- Error 5 - SCTL=CONSOLE.ACK failed to skip with CONSOLE ACK set.
- Error 6 - SCTL=CONSOLE.ATTN skipped with CONSOLE ATTN clear.
- Error 7 - SCTL=CONSOLE.ACK skipped with CONSOLE ACK clear.
- Error 8 - SCTL=CONSOLE.ATTN failed to skip with CONSOLE ATTN set.

:14299
:14300
:14301
:14302
:14303
:14304
:14305
:14306
:14307
:14308
:14309
:14310
:14311
:14312
:14313
:14314
:14315
:14316
:14317
:14318
:14319
:14320
:14321
:14322
:14323
:14324
:14325
:14326
:14327
:14328
:14329
:14330
:14331
:14332
:14333
:14334
:14335
:14336
:14337
:14338
:14339
:14340
:14341
:14342
:14343
:14344
:14345
:14346
:14347
:14348
:14349
:14350
:14351
:14352
:14353

: Debug:

Error 1 - This error indicates a problem with the MUX & INC CTL PAL or its inputs. The inputs to this PAL during the skip instruction should be as follows: ALU N H and ALU V H should both be high, CSR 4-0 should equal 1C(H) and JUMP INSTR L should be high. A problem with any input probably indicates a broken etch. If the inputs are OK, then check for a high out of both output pins (pins 19 and 20). A low on either one indicates a bad PAL. A less likely possibility is that the SEQ.CTL PAL is driving pin 5 of the 3 input negated 'OR' gate low, causing a skip.

Error 2 - Same as error 1 except ALU N H and ALU V H should both be low at the input to the PAL.

Error 3 - This error indicates that a skip did not occur when it should of. First check the inputs to the MUX & INCR CTL PAL during the skip instruction for the following: ALU N H should be high, ALU V H should be low, CSR 4-0 should equal 1C(H), and JUMP INSTR L should be high. If the inputs are OK, check the output for a low on pin 12. If the output is wrong, suspect the MUX & INCR CTL PAL. Otherwise check pin 4 of the three input negated 'OR' gate for a low. If pin 4 is low, the 'OR' gate is probably bad.

Error 4 - Same as error 3 except ALU N H should be low and ALU V H should be high going into the PAL.

Error 5 - This error indicates a problem with the signal CONSOLE ACK, the MUX & INCR CTL PAL, or the three input negated 'OR' gate. The inputs to the MUX & INCR CTL PAL during the skip instruction should be as follows: CONSOLE ACK L should be low, CSR 4-0 should equal 19(H), and JUMP INSTR L should be high. CONSOLE ACK L comes from the 8 D latch used in the interrupt logic. Check the input (pin 8) for a low and the output (pin 9) for a low. If the input is OK, check that the clock input on pin 11 is working. The signal CONS ACK L at the input comes from the 8 bit latch fed by the console logic. Check for a low at pin 9 of this IC. If incorrect, check for a low on the enable (pin 14) and a high on the reset (pin 15). If these are OK, then check for a low on SEL 1 and SEL 2 (pins 1 and 2) and a high on SEL 4 (pin 3) during the time that the console is writing this latch. If the inputs to the MUX & INCR CTL PAL are OK, check output pin 19 for a low during the skip instruction. The PAL is bad if this output is high. Otherwise check for a low on pin 3 of the three input negated 'OR' gate. The 'OR' gate is bad if this input is OK.

Error 6 - This error indicates a problem with the signal CONSOLE ATTN or the MUX & INCR CTL PAL. The inputs to the MUX & INCR CTL PAL during the skip instruction should be as follows: CONSOLE ATTN L should be high, CSR 4-0 should equal 18(H), and JUMP INSTR L should be high. CONSOLE ATTN L comes from the 8 D latch used in the interrupt logic. Check the input (pin 13) for a high and the output (pin 12) for a high. If the input is OK, check that the clock input on pin 11 is working. The signal CONS ATTN L at the input comes from the 8 bit latch fed by the console logic. Check for a high at pin 10


```

14354 : of this IC. If incorrect, check for a low on the enable (pin 14) and
14355 : a high on the reset (pin 15). If these are OK, then check for a high
14356 : on SEL 1 and SEL 4 (pins 1 and 3) and a low on SEL 2 (pin 2) during
14357 : the time that the console is writing this latch. If the inputs to
14358 : the MUX & INCR CTL PAL are OK, check output pin 19 for a high during
14359 : the skip instruction. The PAL is bad if this output is low. A less
14360 : likely possibility is the the SEQ.CTL PAL is driving pin 5 of the
14361 : three input negated 'OR' gate low, causing a skip.
14362 :
14363 : Error 7 - The inputs are the same as error 5 except CONSOLE ACK L
14364 : should be high and the output of the MUX & INCR CTL PAL on pin 19
14365 : should be high.
14366 :
14367 : Error 8 - same as error 6 except CONSOLE ATTN L should be low and the
14368 : output of the MUX & INCR CTL PAL on pin 19 should be low.
14369 :
14370 :
14371 : T.1F:
U 3599, B65E,15 :14372 MOV LS[BEGIN.TEST] TO WR[0] ;SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
U 359A, 3E80,15 :14373 MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
:14374 : ;BIT15 INDICATES START OF TEST TO CONSOLE
U 359B, 10E0,15 :14375 MISC [SET.CP.ATTN] ;ISSUE CPU ATTN TO CONSOLE
:14376 :
WAIT.T1F.0:
U 359C, 8B59,C4 :14377 JMP [WAIT.T1F.0] ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
U 359D, E58A,15 :14378 CLR LS[ERROR.MASK] ;CLEAR ERROR MASK.
U 359E, 65A0,15 :14379 CLR LS[PREVIOUS.ERROR] ;CLEAR PREVIOUS ERROR NUMBER
U 359F, 2F80,15 :14380 CLR WR[0]
U 35A0, 2040,15 :14381 INC WR[0] ;SET WRO TO 1
U 35A1, BE82,15 :14382 MOV WR[0] TO LS[ERROR.NUMBER] ;SET ERROR NUMBER TO FIRST ERROR
U 35A2, A3C0,15 :14383 ROL WR[0] ;SET WRO TO 2
U 35A3, 3E8C,15 :14384 MOV WR[0] TO LS[MODULE.NUM] ;SET UP MODULE CODE FOR CPU MODULE
U 35A4, B66E,15 :14385 MOV LS[BIT23] TO WR[0]
U 35A5, 3E80,15 :14386 MOV WR[0] TO LS[CONTROL.STATUS] ;BIT23 = EXPECTED AND RECEIVED DATA N/A
:14387 :
LOOP.T1F.1:
U 35A6, 2F82,95 :14388 CLR WR[1]
U 35A7, BFFC,95 :14389 MOV WR[1] TO LS[ALU.CC] ;CLEAR ALL ALU CODES
U 35A8, DB00,1C :14390 SKIP,IF[LS]
U 35A9, 0B5A,E4 :14391 JMP [T1F.2] ;JUMP OVER ERROR CALL
U 35AA, 2F80,15 :14392 CLR WR[0]
U 35AB, 369E,95 :14393 MOV LS[ONES] TO WR[1]
U 35AC, 0869,3C :14394 JSR [CHECK.RESULT] ;CALL ERROR ROUTINE WITH DELIBERATELY
:14395 : ;ERRONEOUS DATA TO FLAG SKIP ERROR
U 35AD, 8B5A,64 :14396 JMP [LOOP.T1F.1] ;LOOP ON ERROR IF ENABLED
:14397 :
T1F.2:
U 35AE, 8870,5C :14398 JSR [INC.EN] ;INCREMENT ERROR NUMBER TO 2
:14399 :
LOOP.T1F.2:
U 35AF, B642,95 :14400 MOV LS[BIT1] TO WR[1]
U 35B0, 4746,95 :14401 BIS LS[BIT3] TO WR[1]
U 35B1, BFFC,95 :14402 MOV WR[1] TO LS[ALU.CC] ;SET ALU N AND ALU V
U 35B2, DB00,1C :14403 SKIP,IF[LS]
U 35B3, 8B5B,84 :14404 JMP [T1F.3] ;JUMP OVER ERROR CALL
U 35B4, 2F80,15 :14405 CLR WR[0]
U 35B5, 369E,95 :14406 MOV LS[ONES] TO WR[1]
U 35B6, 0869,3C :14407 JSR [CHECK.RESULT] ;CALL ERROR ROUTINE WITH DELIBERATELY
:14408 : ;ERRONEOUS DATA TO FLAG SKIP ERROR
  
```

```

U 35B7, 8B5A,F4 :14409      JMP [LOOP.T1F.2]          :LOOP ON ERROR IF ENABLED
:14410
U 35B8, 8870,5C :14411      T1F.3: JSR [INC.EN]          :INCREMENT ERROR NUMBER TO 3
:14412      LOOP.T1F.3:
U 35B9, 3646,95 :14413      MOV LS[BIT3] TO WR[1]
U 35BA, BFFC,95 :14414      MOV WR[1] TO LS[ALU.CC]      :SET ALU N
U 35BB, DB00,1C :14415      SKIP.IF[LSS]
U 35BC, 5B00,1E :14416      SKIP          :ERROR -- SKIP NEXT INSTRUCTION
U 35BD, 0B5C,24 :14417      JMP [T1F.4]          :GO TO NEXT SECTION
U 35BE, 2F80,15 :14418      CLR WR[0]
U 35BF, 369E,95 :14419      MOV LS[ONES] TO WR[1]
U 35C0, 0869,3C :14420      JSR [CHECK.RESULT]      :CALL ERROR ROUTINE WITH DELIBERATELY
:14421      : ERRONEOUS DATA TO FLAG SKIP ERROR
U 35C1, 0B5B,94 :14422      JMP [LOOP.T1F.3]      :LOOP ON ERROR IF ENABLED
:14423
U 35C2, 8870,5C :14424      T1F.4: JSR [INC.EN]          :INCREMENT ERROR NUMBER TO 4
:14425      LOOP.T1F.4:
U 35C3, 3646,95 :14426      MOV LS[BIT3] TO WR[1]
U 35C4, BFFC,95 :14427      MOV WR[1] TO LS[ALU.CC]      :SET ALU N
U 35C5, DB00,1C :14428      SKIP.IF[LSS]
U 35C6, 5B00,1E :14429      SKIP          :ERROR -- SKIP NEXT INSTRUCTION
U 35C7, 8B5C,C4 :14430      JMP [T1F.5]          :GO TO NEXT SECTION
U 35C8, 2F80,15 :14431      CLR WR[0]
U 35C9, 369E,95 :14432      MOV LS[ONES] TO WR[1]
U 35CA, 0869,3C :14433      JSR [CHECK.RESULT]      :CALL ERROR ROUTINE WITH DELIBERATELY
:14434      : ERRONEOUS DATA TO FLAG SKIP ERROR
U 35CB, 8B5C,34 :14435      JMP [LOOP.T1F.4]      :LOOP ON ERROR IF ENABLED
:14436
U 35CC, 8870,5C :14437      T1F.5: JSR [INC.EN]          :INCREMENT ERROR NUMBER TO 5
U 35CD, B660,95 :14438      MOV LS[BIT16] TO WR[1]
U 35CE, C76A,95 :14439      BIS LS[BIT21] TO WR[1]
U 35CF, BE80,95 :14440      MOV WR[1] TO LS[CONTROL.STATUS]
U 35D0, 10E0,15 :14441      MISC [SET.CP.ATTN]
:14442      WAIT.T1F.5:
U 35D1, 8B5D,14 :14443      JMP [WAIT.T1F.5]          :LOOP WAITING FOR CONSOLE TO RESPOND
U 35D2, 366E,95 :14444      MOV LS[BIT23] TO WR[1]
U 35D3, BE80,95 :14445      MOV WR[1] TO LS[CONTROL.STATUS] :RESTORE CONTROL AND STATUS WORD.
:14446      LOOP.T1F.5:
U 35D4, DB00,19 :14447      SKIP.IF[CONSOLE.ACK]
U 35D5, 5B00,1E :14448      SKIP          :ERROR -- SKIP NEXT INSTRUCTION
U 35D6, 8B5D,B4 :14449      JMP [T1F.6]          :GO TO NEXT SECTION
U 35D7, 2F80,15 :14450      CLR WR[0]
U 35D8, 369E,95 :14451      MOV LS[ONES] TO WR[1]
U 35D9, 0869,3C :14452      JSR [CHECK.RESULT]      :CALL ERROR ROUTINE WITH DELIBERATELY
:14453      : ERRONEOUS DATA TO FLAG SKIP ERROR
U 35DA, 8B5D,44 :14454      JMP [LOOP.T1F.5]      :LOOP ON ERROR IF ENABLED
:14455
U 35DB, 8870,5C :14456      T1F.6: JSR [INC.EN]          :INCREMENT ERROR NUMBER TO 6
:14457      LOOP.T1F.6:
U 35DC, 5B00,18 :14458      SKIP.IF[CONSOLE.ATTN]
U 35DD, 8B5E,24 :14459      JMP [T1F.7]
U 35DE, 2F80,15 :14460      CLR WR[0]
U 35DF, 369E,95 :14461      MOV LS[ONES] TO WR[1]
U 35E0, 0869,3C :14462      JSR [CHECK.RESULT]      :CALL ERROR ROUTINE WITH DELIBERATELY
:14463      : ERRONEOUS DATA TO FLAG SKIP ERROR
  
```



```

U 35E1, 0B5D,C4 :14464      JMP [LOOP.T1F.6]                ;LOOP ON ERROR IF ENABLED
                  :14465
U 35E2, 8870,5C :14466      T1F.7: JSR [INC.EN]                ;INCREMENT ERROR NUMBER TO 7
U 35E3, B660,95 :14467      MOV LS[BIT16] TO WR[1]
U 35E4, 4768,95 :14468      BIS LS[BIT20] TO WR[1]
U 35E5, BE80,95 :14469      MOV WR[1] TO LS[CONTROL.STATUS]
U 35E6, 10E0,15 :14470      MISC [SET.CP.ATTN]
                  :14471
U 35E7, 8B5E,74 :14472      WAIT.T1F.7: JMP [WAIT.T1F.7]          ;LOOP WITING FOR CONSOLE TO RESPOND
U 35E8, 366E,95 :14473      MOV LS[BIT23] TO WR[1]
U 35E9, BE80,95 :14474      MOV WR[1] TO LS[CONTROL.STATUS] ;RESTORE CONTROL AND STATUS WORD.
                  :14475
U 35EA, DB00,19 :14476      LOOP.T1F.7: SKIP.IF[CONSOLE.ACK]
U 35EB, 8B5F,04 :14477      JMP [T1F.8]                ;JUMP OVER ERROR CALL
U 35EC, 2F80,15 :14478      CLR WR[0]
U 35ED, 369E,95 :14479      MOV LS[ONES] TO WR[1]
U 35EE, 0869,3C :14480      JSR [CHECK.RESULT]          ;CALL ERROR ROUTINE WITH DELIBERATELY
                  :14481      ; ERRONEOUS DATA TO FLAG SKIP ERROR
U 35EF, 0B5E,A4 :14482      JMP [LOOP.T1F.7]          ;LOOP ON ERROR IF ENABLED
                  :14483
U 35F0, 8870,5C :14484      T1F.8: JSR [INC.EN]                ;INCREMENT ERROR NUMBER TO 8
                  :14485
U 35F1, 5B00,18 :14486      LOOP.T1F.8: SKIP.IF[CONSOLE.ATTN]
U 35F2, 5B00,1E :14487      SKIP
                  :14488      ;ERROR -- SKIP NEXT INSTRUCTION
U 35F3, 0B5F,84 :14488      JMP [T1F.8.B]            ;GO TO NEXT SECTION
U 35F4, 2F80,15 :14489      CLR WR[0]
U 35F5, 369E,95 :14490      MOV LS[ONES] TO WR[1]
U 35F6, 0869,3C :14491      JSR [CHECK.RESULT]          ;CALL ERROR ROUTINE WITH DELIBERATELY
                  :14492      ; ERRONEOUS DATA TO FLAG SKIP ERROR
U 35F7, 0B5F,14 :14493      JMP [LOOP.T1F.8]          ;LOOP ON ERROR IF ENABLED
                  :14494
U 35F8, B660,95 :14495      T1F.8.B: MOV LS[BIT16] TO WR[1]
U 35F9, BE80,95 :14496      MOV WR[1] TO LS[CONTROL.STATUS]
U 35FA, 10E0,15 :14497      MISC [SET.CP.ATTN]
                  :14498
U 35FB, 0B5F,B4 :14499      WAIT.T1F.END: JMP [WAIT.T1F.END]      ;LOOP WITING FOR CONSOLE TO RESPOND
                  :14500
END.T1F:
  
```

Page "TEST 20 - STATE REG and skip test (M8390 CPU Module)"

++

Test Description:

This test checks the STATE REG which is set up with a MISC instruction. The STATE REG can only be checked by setting it up and executing a SKIP condition that uses the STATE REG. There are 4 skip conditions to be tested: skip if state 0 clear, skip if state 0 set, skip if state 1 clear, and skip if state 1 set. There are 7 codes of the MISC instruction that set up various STATE REG values. The test will set up a value in the STATE REG using a MISC instruction. Then instructions are executed with the SCTL field set to skip on various state values. A minimum of 2 and a maximum of 4 instructions using the STATE REG for SCTL will be executed.

Assumptions:

It is assumed that the previous skip test using the MUX & INCR CTL PAL has run successfully.

Test Steps:

- 1) Set up error mask, error number, and module number in LS (for error printouts) and clear previous error number in LS.
- 2) Issue a MISC instruction to clear STATE REG <1:0>.
- 3) Issue a MISC instruction to set STATE REG 0.
- 4) Issue a NOP with SCTL=STATE.ZERO.SET. Check for skip.
- 5) Issue a NOP with SCTL=STATE.ONE.SET. Check for no skip.
- 6) Issue a NOP with SCTL=STATE.ZERO.CLR. Check for no skip.
- 7) Issue a NOP with SCTL=STATE.ONE.CLR. Check for skip.
- 8) Issue a MISC instruction to set STATE REG 1, clear STATE REG 0.
- 9) Repeat steps 4 through 7 checking for no skip, skip, skip, no skip respectively.
- 10) Issue a MISC instruction to clear STATE REG 1, set STATE REG 0.
- 11) Repeat steps 4 and 7 checking for skip in both cases.
- 12) Issue a MISC instruction to set STATE REG 1 (now both set).
- 13) Repeat steps 4 and 5 checking for skip in both cases.
- 14) Issue a MISC instruction to clear STATE REG 1 and STATE REG 0 (one instruction: MISC 1 code 0).
- 15) Repeat steps 6 and 7 checking for skip in both cases.
- 16) Issue 2 MISC instructions, one to set STATE REG 1 and one to set STATE REG 0.
- 17) Issue a MISC instruction to clear STATE REG 1 (now 0 set, 1 clear).
- 18) Repeat steps 4 and 7 checking for skip in both cases.
- 19) Issue 2 MISC instructions, one to set STATE REG 1 and one to set STATE REG 0.
- 20) Issue a MISC instruction to clear STATE REG 0 (now 1 set, 0 clear).
- 21) Repeat steps 5 and 6 checking for skip in both cases.
- 22) Issue CPU ATTN with bit 12 set to cause the console to disable memory references.
- 23) Issue 2 MISC instructions, one to set STATE REG 1 and one to set STATE REG 0.
- 24) Issue a DECODE instruction to cause IRD STATE L to be asserted by setting bit 4 (OPC/SPEC) and making IFUNC=1. This should clear

:14556
:14557
:14558
:14559
:14560
:14561
:14562
:14563
:14564
:14565
:14566
:14567
:14568
:14569
:14570
:14571
:14572
:14573
:14574
:14575
:14576
:14577
:14578
:14579
:14580
:14581
:14582
:14583
:14584
:14585
:14586
:14587
:14588
:14589
:14590
:14591
:14592
:14593
:14594
:14595
:14596
:14597
:14598
:14599
:14600
:14601
:14602
:14603
:14604
:14605
:14606
:14607
:14608
:14609
:14610

both state regs.
25) Repeat steps 6 and 7 checking for skip in both cases.
26) Issue CPU ATTN with bit 13 set to cause the console to enable memory references (for housekeeping purposes).

Errors:

- Error 1 - SCTL=STATE.ZERO.SET failed to skip with state 0 set and state 1 clear.
- Error 2 - SCTL=STATE.ONE.SET skipped with state 0 set and state 1 clear.
- Error 3 - SCTL=STATE.ZERO.CLR skipped with state 0 set and state 1 clear.
- Error 4 - SCTL=STATE.ONE.CLR failed to skip with state 0 set and state 1 clear.
- Error 5 - SCTL=STATE.ZERO.SET skipped with state 0 clear and state 1 set.
- Error 6 - SCTL=STATE.ONE.SET failed to skip with state 0 clear and state 1 set.
- Error 7 - SCTL=STATE.ZERO.CLR failed to skip with state 0 clear and state 1 set.
- Error 8 - SCTL=STATE.ONE.CLR skipped with state 0 clear and state 1 set.
- Error 9 - Clear state reg 1, set state reg 0 failed to set state 0.
- Error A - Clear state reg 1, set state reg 0 failed to clear state 1.
- Error B - Set state reg 1 cleared state reg 0.
- Error C - Set state reg 1 failed to set state 1.
- Error D - Clear state reg <1:0> failed to clear state 0.
- Error E - Clear state reg <1:0> failed to clear state 1.
- Error F - Clear state reg 1 cleared state reg 0.
- Error 10- Clear state reg 1 failed to clear state 1.
- Error 11- Clear state reg 0 cleared state reg 1.
- Error 12- Clear state reg 0 failed to clear state 0.
- Error 13- Asserting IRD STATE L failed to clear state reg 0.
- Error 14- Asserting IRD STATE L failed to clear state reg 1.

Debug:

- Error 1 - This error could be caused by a problem with either the state reg or the MUX & INCR CTL PAL. Check the state reg inputs to the MUX & INCR CTL PAL for a high on STATE 0 H and a low on STATE 1 H during the NOP with SCTL=STATE.ZERO.SET. The other inputs to the PAL were used before. If these inputs are OK, then suspect the PAL itself. Otherwise check STATE 1 H and STATE 0 H coming out of the STATE REG. MISC DECODER PAL. If the outputs are incorrect, check the inputs for the following during the MISC function that sets state reg 0: MISC INSTR H should be high, CSR 15-12 should equal 5(H), and IRD STATE L should be high. If these are correct, then the STATE REG. MISC DECODER PAL is suspect.
- Error 2 - See error 1. The inputs are the same, but this instruction depends on state 1 being clear, whereas error 1 depended on state 0 only. Check state 1 for a low, and if incorrect, check the STATE REG. MISC DECODER PAL during the MISC instruction that clears STATE 1 and STATE 0. The inputs to this PAL are the same as above except

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 20 - STATE REG 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 20 - STATE REG and skip test (M8390 CPU Module)

F 9
Fiche 2 Frame F9
ENKCA Micro-diagnostic for DAP module

Sequence 315
Rev 01.00 Page 309

:14611 : CSR 15-12 should equal 0(H).
:14612 :
:14613 : Error 3 - If this is the first error, suspect the MUX & INCR CTL PAL
:14614 : itself.
:14615 :
:14616 : Error 4 - Same as error 3.
:14617 :
:14618 : Error 5 - This error could be caused by a problem with either the
:14619 : state reg or the MUX & INCR CTL PAL. Check the state reg inputs to
:14620 : the MUX & INCR CTL PAL for a low on STATE 0 H and a high on STATE 1
:14621 : H during the NOP with SCTL=STATE.ZERO.SET. The other inputs to the
:14622 : PAL were used before. If these inputs are OK, then suspect the PAL
:14623 : itself. Otherwise check STATE 1 H and STATE 0 H coming out of the
:14624 : STATE REG. MISC DECODER PAL. If the outputs are incorrect, check
:14625 : the inputs for the following during the MISC function that sets state
:14626 : reg 1, clear state reg 0: MISC INSTR H should be high, CSR 15-12
:14627 : should equal 2(H), and IRD STATE L should be high. If these are
:14628 : correct, then the STATE REG. MISC DECODER PAL is suspect.
:14629 :
:14630 : Error 6 - See error 1. The inputs are the same, but this instruction
:14631 : depends on state 1 being clear, whereas error 1 depended on state
:14632 : 0 only. Check state 1 for a high, and if incorrect, check the STATE
:14633 : REG. MISC DECODER PAL as in error 5 above.
:14634 :
:14635 : Error 7 - If this is the first error, suspect the MUX & INCR CTL PAL
:14636 : itself.
:14637 :
:14638 : Error 8 - Same as error 7.
:14639 :
:14640 : Error 9 - Suspect the STATE REG. MISC DECODER PAL. The inputs are as
:14641 : follows during the instruction that clears state reg 1, sets state reg
:14642 : 0: MISC INSTR H should be high, CSR 15-12 should equal 1(H), and
:14643 : IRD STATE L should be high. STATE 1 H should be low, STATE 0 H should
:14644 : be high.
:14645 :
:14646 : Error A - Same as error 9.
:14647 :
:14648 : Error B - Suspect the STATE REG. MISC DECODER PAL. The inputs are as
:14649 : follows during the instruction that sets state reg 1: MISC INSTR H
:14650 : should be high, CSR 15-12 should equal 6(H), and IRD STATE L should
:14651 : be high. STATE 1 H should be high, STATE 0 H should be low (unchanged).
:14652 :
:14653 : Error C - Same as error B
:14654 :
:14655 : Error D - Suspect the STATE REG. MISC DECODER PAL. The inputs are as
:14656 : follows during the instruction that clears state reg 1 and state reg
:14657 : 0: MISC INSTR H should be high, CSR 15-12 should equal 0(H), and
:14658 : IRD STATE L should be high. STATE 1 H should be low, STATE 0 H should
:14659 : be low.
:14660 :
:14661 : Error E - Same as error D.
:14662 :
:14663 : Error F - Suspect the STATE REG. MISC DECODER PAL. The inputs are as
:14664 : follows during the instruction that clears state reg 1: MISC INSTR H
:14665 : should be high, CSR 15-12 should equal 4(H), and IRD STATE L should

ZZ-ENKCA-1.0
: ENKCA.MCR
: ENKCA.MIC

: ENKCA.MIC

TEST 20 - STATE REG 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
TEST 20 - STATE REG and skip test (M8390 CPU Module)

G 9

ENKCA Micro-diagnostic for DAP module
CPU Module)

Fiche 2 Frame G9

Sequence 316
Rev 01.00

Page 310

:14666 : be high. STATE 1 H should be low, STATE 0 H should be high (unchanged).
:14667 :
:14668 : Error 10 - Same as error F.
:14669 :
:14670 : Error 11 - Suspect the STATE REG. MISC DECODER PAL. The inputs are as
:14671 : follows during the instruction that clears state reg 0: MISC INSTR H
:14672 : should be high, CSR 15-12 should equal 3(H), and IRD STATE L should
:14673 : be high. STATE 0 H should be low, STATE 1 H should be high (unchanged).
:14674 :
:14675 : Error 12 - Same as error 11.
:14676 :
:14677 : Error 13 - This error could be caused by the STATE REG. MISC DECODER
:14678 : PAL failing or IRD STATE L failing. The only input to the STATE REG
:14679 : PAL that matters is IRD STATE L. Check for a low on the signal during
:14680 : the DECODE instruction. The low should force both STATE 0 and STATE 1
:14681 : low. If this signal is OK, suspect the STATE REG PAL, else check IRD
:14682 : STATE L coming out of the CM.RDEST PAL for a low. If incorrect check
:14683 : the inputs to the CM.RDEST PAL for DECODE INSTR H, CSR 04 H, and CSR
:14684 : 09 H all high. If these are OK, the CM.RDEST PAL is bad. DECODE INSTR
:14685 : H comes from the LS CONTROL PAL. If this signal is low at the output,
:14686 : check the inputs for lows on CSR 22-19.
:14687 :
:14688 : Error 14 - See error 13. The STATE REG. MISC DECODER PAL is the most
:14689 : likely suspect.
:14690 :
:14691 :
:14692 :
:14693 :
:14694 :
:14695 :
:14696 :
:14697 :
:14698 :
:14699 :
:14700 :
:14701 :
:14702 :
:14703 :
:14704 :
:14705 :
:14706 :
:14707 :
:14708 :
:14709 :
:14710 :
:14711 :
:14712 :
:14713 :
:14714 :
:14715 :
:14716 :
:14717 :
:14718 :
:14719 :
:14720 :

T.20:

U 35FC, B65E,15 :14693
U 35FD, 3E80,15 :14694
U 35FE, 10E0,15 :14695
U 35FF, 8B5F,F4 :14696
U 3600, E58A,15 :14697
U 3601, 65A0,15 :14698
U 3602, 2F80,15 :14699
U 3603, 2040,15 :14700
U 3604, BE82,15 :14701
U 3605, A3C0,15 :14702
U 3606, 3E8C,15 :14703
U 3607, B66E,15 :14704
U 3608, 3E80,15 :14705
U 3609, 9000,15 :14706
U 360A, 9050,15 :14707
U 360B, 5B00,0C :14708
U 360C, 5B00,1E :14709
U 360D, 8B61,24 :14710
U 360E, 2F80,15 :14711
U 360F, 369E,95 :14712
U 3610, 0869,3C :14713
U 3611, 0B60,B4 :14714
U 3612, 8870,5C :14715

MOV LS[BEGIN.TEST] TO WR[0] :SET BIT15 IN WRO FOR CONTROL AND STATUS WORD
MOV WR[0] TO LS[CONTROL.STATUS] :SET BIT15 IN CNTROL AND STATUS WORD
MISC [SET.CP.ATTN] :BIT15 INDICATES START OF TEST TO CONSOLE
ISSUE CPU ATTN TO CONSOLE
WAIT.T20.0:
JMP [WAIT.T20.0] :LOOP HERE WAITING FOR CONSOLE TO RESPOND
CLR LS[ERROR.MASK] :CLEAR ERROR MASK.
CLR LS[PREVIOUS.ERROR] :CLEAR PREVIOUS ERROR NUMBER
CLR WR[0]
INC WR[0] :SET WRO TO 1
MOV WR[0] TO LS[ERROR.NUMBER] :SET ERROR NUMBER TO FIRST ERROR
ROL WR[0] :SET WRO TO 2
MOV WR[0] TO LS[MODULE.NUM] :SET UP MODULE CODE FOR CPU MODULE
MOV LS[BIT23] TO WR[0]
MOV WR[0] TO LS[CONTROL.STATUS] :BIT23 = EXPECTED AND RECEIVED DATA N/A
MISC [CLR] :CLEAR STATE REG
MISC [SET.S0] :SET STATE REG 0
LOOP.T20.1:
SKIP.IF[STATE.ZERO.SET]
SKIP :ERROR -- SKIP NEXT INSTRUCTION
JMP [T20.2] :GO ON...
CLR WR[0]
MOV LS[ONES] TO WR[1]
JSR [CHECK.RESULT] :CALL ERROR ROUTINE WITH DELIBERATELY
:ERRONEOUS DATA TO FLAG SKIP ERROR
JMP [LOOP.T20.1] :LOOP ON ERROR IF ENABLED
T20.2:
JSR [INC.EN] :INCREMENT ERROR NUMBER TO 2

```

:14721 LOOP.T20.2:
U 3613, DB00.0D :14722     SKIP IF[STATE.ONE.SET]
U 3614, 0B61.94 :14723     JMP [T20.3]
U 3615, 2F80.15 :14724     CLR WR[0]
U 3616, 369E.95 :14725     MOV LS[ONES] TO WR[1]
U 3617, 0B69.3C :14726     JSR [CHECK.RESULT]
:14727
:14728     JMP [LOOP.T20.2]
U 3618, 0B61.34 :14729
:14730 T20.3:
U 3619, 8870.5C :14731     JSR [INC.EN]
:14732 LOOP.T20.3:
U 361A, DB00.0E :14733     SKIP IF[STATE.ZERO.CLR]
U 361B, 0B62.04 :14734     JMP [T20.4]
U 361C, 2F80.15 :14735     CLR WR[0]
U 361D, 369E.95 :14736     MOV LS[ONES] TO WR[1]
U 361E, 0B69.3C :14737     JSR [CHECK.RESULT]
:14738     JMP [LOOP.T20.3]
U 361F, 0B61.A4 :14739
:14740 T20.4:
U 3620, 8870.5C :14741     JSR [INC.EN]
:14742 LOOP.T20.4:
U 3621, 5B00.0F :14743     SKIP IF[STATE.ONE.CLR]
U 3622, 5B00.1E :14744     SKIP
U 3623, 8B62.84 :14745     JMP [T20.5]
U 3624, 2F80.15 :14746     CLR WR[0]
U 3625, 369E.95 :14747     MOV LS[ONES] TO WR[1]
U 3626, 0B69.3C :14748     JSR [CHECK.RESULT]
:14749     JMP [LOOP.T20.4]
U 3627, 8B62.14 :14750
:14751 T20.5:
U 3628, 8870.5C :14752     JSR [INC.EN]
U 3629, 9000.15 :14753     MISC [CLR]
U 362A, 9060.15 :14754     MISC [SET.S1]
:14755 LOOP.T20.5:
U 362B, 5B00.0C :14756     SKIP IF[STATE.ZERO.SET]
U 362C, 0B63.14 :14757     JMP [T20.6]
U 362D, 2F80.15 :14758     CLR WR[0]
U 362E, 369E.95 :14759     MOV LS[ONES] TO WR[1]
U 362F, 0B69.3C :14760     JSR [CHECK.RESULT]
:14761     JMP [LOOP.T20.5]
U 3630, 8B62.B4 :14762
:14763 T20.6:
U 3631, 8870.5C :14764     JSR [INC.EN]
:14765 LOOP.T20.6:
U 3632, DB00.0D :14766     SKIP IF[STATE.ONE.SET]
U 3633, 5B00.1E :14767     SKIP
U 3634, 8B63.94 :14768     JMP [T20.7]
U 3635, 2F80.15 :14769     CLR WR[0]
U 3636, 369E.95 :14770     MOV LS[ONES] TO WR[1]
U 3637, 0B69.3C :14771     JSR [CHECK.RESULT]
:14772     JMP [LOOP.T20.6]
U 3638, 0B63.24 :14773
:14774 T20.7:
U 3639, 8870.5C :14775     JSR [INC.EN]
:14776 LOOP.T20.7:

```

;CALL ERROR ROUTINE WITH DELIBERATELY
 ; ERRONEOUS DATA TO FLAG SKIP ERROR
 ;LOOP ON ERROR IF ENABLED

;INCREMENT ERROR NUMBER TO 3

;CALL ERROR ROUTINE WITH DELIBERATELY
 ; ERRONEOUS DATA TO FLAG SKIP ERROR
 ;LOOP ON ERROR IF ENABLED

;INCREMENT ERROR NUMBER TO 4

;ERROR -- SKIP NEXT INSTRUCTION
 ;GO ON...

;CALL ERROR ROUTINE WITH DELIBERATELY
 ; ERRONEOUS DATA TO FLAG SKIP ERROR
 ;LOOP ON ERROR IF ENABLED

;INCREMENT ERROR NUMBER TO 5

;SET STATE REG 1

;GO ON...

;CALL ERROR ROUTINE WITH DELIBERATELY
 ; ERRONEOUS DATA TO FLAG SKIP ERROR
 ;LOOP ON ERROR IF ENABLED

;INCREMENT ERROR NUMBER TO 6

;ERROR -- SKIP NEXT INSTRUCTION

;CALL ERROR ROUTINE WITH DELIBERATELY
 ; ERRONEOUS DATA TO FLAG SKIP ERROR
 ;LOOP ON ERROR IF ENABLED

;INCREMENT ERROR NUMBER TO 7

ZZ-ENKCA-1.0 : ENKCA.MIC
: ENKCA.MCR
: ENKCA.MIC

I 9
TEST 20 - STATE REG 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
TEST 20 - STATE REG and skip test (M8390 CPU Module)
Fiche 2 Frame 19
Sequence 318
Rev 01.00 Page 312

```
U 363A, DB00,0E :14776      SKIP.IF[STATE.ZERO.CLR]
U 363B, 5B00,1E :14777      SKIP
U 363C, 8B64,14 :14778      JMP [T20.8]
U 363D, 2F80,15 :14779      CLR WR[0]
U 363E, 369E,95 :14780      MOV LS[ONES] TO WR[1]
U 363F, 0B69,3C :14781      JSR [CHECK.RESULT]
                                :CALL ERROR ROUTINE WITH DELIBERATELY
                                : ERRONEOUS DATA TO FLAG SKIP ERROR
                                :LOOP ON ERROR IF ENABLED
U 3640, 8B63,A4 :14782      JMP [LOOP.T20.7]
U 3641, 8870,5C :14783      T20.8: JSR [INC.EN]
                                :INCREMENT ERROR NUMBER TO 8
                                :GO ON...
U 3642, 5B00,0F :14784      LOOP.T20.8: SKIP.IF[STATE.ONE.CLR]
U 3643, 8B64,84 :14785      JMP [T20.9]
U 3644, 2F80,15 :14786      CLR WR[0]
U 3645, 369E,95 :14787      MOV LS[ONES] TO WR[1]
U 3646, 0B69,3C :14788      JSR [CHECK.RESULT]
                                :CALL ERROR ROUTINE WITH DELIBERATELY
                                : ERRONEOUS DATA TO FLAG SKIP ERROR
                                :LOOP ON ERROR IF ENABLED
U 3647, 8B64,24 :14789      T20.9: JMP [LOOP.T20.8]
U 3648, 8870,5C :14790      JSR [INC.EN]
U 3649, 1010,15 :14791      MISC [CLR.S1&SET.S0]
                                :INCREMENT ERROR NUMBER TO 9
                                :CLEAR S1 SET S0
U 364A, 5B00,0C :14792      LOOP.T20.9: SKIP.IF[STATE.ZERO.SET]
U 364B, 5B00,1E :14793      SKIP
U 364C, 0B65,14 :14794      JMP [T20.A]
U 364D, 2F80,15 :14795      CLR WR[0]
U 364E, 369E,95 :14796      MOV LS[ONES] TO WR[1]
U 364F, 0B69,3C :14797      JSR [CHECK.RESULT]
                                :CALL ERROR ROUTINE WITH DELIBERATELY
                                : ERRONEOUS DATA TO FLAG SKIP ERROR
                                :LOOP ON ERROR IF ENABLED
U 3650, 0B64,A4 :14798      T20.A: JMP [LOOP.T20.9]
U 3651, 8870,5C :14799      JSR [INC.EN]
                                :INCREMENT ERROR NUMBER TO A
                                :GO ON...
U 3652, 5B00,0F :14800      LOOP.T20.A: SKIP.IF[STATE.ONE.CLR]
U 3653, 5B00,1E :14801      SKIP
U 3654, 8B65,94 :14802      JMP [T20.B]
U 3655, 2F80,15 :14803      CLR WR[0]
U 3656, 369E,95 :14804      MOV LS[ONES] TO WR[1]
U 3657, 0B69,3C :14805      JSR [CHECK.RESULT]
                                :CALL ERROR ROUTINE WITH DELIBERATELY
                                : ERRONEOUS DATA TO FLAG SKIP ERROR
                                :LOOP ON ERROR IF ENABLED
U 3658, 0B65,24 :14806      T20.B: JMP [LOOP.T20.A]
U 3659, 8870,5C :14807      JSR [INC.EN]
U 365A, 9050,15 :14808      MISC [SET.S0]
U 365B, 9060,15 :14809      MISC [SET.S1]
                                :INCREMENT ERROR NUMBER TO B
                                :BOTH SET
U 365C, 5B00,0C :14810      LOOP.T20.B: SKIP.IF[STATE.ZERO.SET]
U 365D, 5B00,1E :14811      SKIP
U 365E, 8B66,34 :14812      JMP [T20.C]
U 365F, 2F80,15 :14813      CLR WR[0]
U 3660, 369E,95 :14814      MOV LS[ONES] TO WR[1]
U 3661, 0B69,3C :14815      JSR [CHECK.RESULT]
                                :CALL ERROR ROUTINE WITH DELIBERATELY
                                : ERRONEOUS DATA TO FLAG SKIP ERROR
                                :LOOP ON ERROR IF ENABLED
U 3662, 8B65,C4 :14816      T20.C: JMP [LOOP.T20.B]
```

U 3663, 8870.5C	:14831	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO C
U 3664, DB00.0D	:14832	LOOP.T20.C:	
U 3665, 5B00.1E	:14833	SKIP.[IF[STATE.ONE.SET]	:ERROR -- SKIP NEXT INSTRUCTION
U 3666, 0B66.B4	:14834	SKIP	:GO ON...
U 3667, 2F80.15	:14835	JMP [T20.D]	
U 3668, 369E.95	:14836	CLR WR[0]	
U 3669, 0B69.3C	:14837	MOV LS[ONES] TO WR[1]	
	:14838	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14839		: ERRONEOUS DATA TO FLAG SKIP ERROR
U 366A, 0B66.44	:14840	JMP [LOOP.T20.C]	:LOOP ON ERROR IF ENABLED
U 366B, 8870.5C	:14841	T20.D:	
U 366C, 9000.15	:14842	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO D
	:14843	MISC [CLR]	:CLEAR BOTH
	:14844	LOOP.T20.D:	
U 366D, DB00.0E	:14845	SKIP.[IF[STATE.ZERO.CLR]	
U 366E, 5B00.1E	:14846	SKIP	:ERROR -- SKIP
U 366F, 8B67.44	:14847	JMP [T20.E]	
U 3670, 2F80.15	:14848	CLR WR[0]	
U 3671, 369E.95	:14849	MOV LS[ONES] TO WR[1]	
U 3672, 0B69.3C	:14850	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14851		: ERRONEOUS DATA TO FLAG SKIP ERROR
U 3673, 0B66.D4	:14852	JMP [LOOP.T20.D]	:LOOP ON ERROR IF ENABLED
U 3674, 8870.5C	:14853	T20.E:	
	:14854	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO E
	:14855	LOOP.T20.E:	
U 3675, 5B00.0F	:14856	SKIP.[IF[STATE.ONE.CLR]	
U 3676, 5B00.1E	:14857	SKIP	:ERROR -- SKIP
U 3677, 0B67.C4	:14858	JMP [T20.F]	
U 3678, 2F80.15	:14859	CLR WR[0]	
U 3679, 369E.95	:14860	MOV LS[ONES] TO WR[1]	
U 367A, 0B69.3C	:14861	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14862		: ERRONEOUS DATA TO FLAG SKIP ERROR
U 367B, 0B67.54	:14863	JMP [LOOP.T20.E]	:LOOP ON ERROR IF ENABLED
U 367C, 8870.5C	:14864	T20.F:	
U 367D, 9060.15	:14865	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO F
U 367E, 9050.15	:14866	MISC [SET.S1]	
U 367F, 1040.15	:14867	MISC [SET.S0]	
	:14868	MISC [CLR.S1]	:0 SET 1 CLEAR
	:14869	LOOP.T20.F:	
U 3680, 5B00.0C	:14870	SKIP.[IF[STATE.ZERO.SET]	
U 3681, 5B00.1E	:14871	SKIP	:ERROR -- SKIP NEXT INSTRUCTION
U 3682, 8B68.74	:14872	JMP [T20.10]	:GO ON...
U 3683, 2F80.15	:14873	CLR WR[0]	
U 3684, 369E.95	:14874	MOV LS[ONES] TO WR[1]	
U 3685, 0B69.3C	:14875	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14876		: ERRONEOUS DATA TO FLAG SKIP ERROR
U 3686, 0B68.04	:14877	JMP [LOOP.T20.F]	:LOOP ON ERROR IF ENABLED
U 3687, 8870.5C	:14878	T20.10:	
	:14879	JSR [INC.EN]	:INCREMENT ERROR NUMBER TO 10
	:14880	LOOP.T20.10:	
U 3688, 5B00.0F	:14881	SKIP.[IF[STATE.ONE.CLR]	
U 3689, 5B00.1E	:14882	SKIP	:ERROR -- SKIP NEXT INSTRUCTION
U 368A, 0B68.F4	:14883	JMP [T20.11]	:GO ON...
U 368B, 2F80.15	:14884	CLR WR[0]	
U 368C, 369E.95	:14885	MOV LS[ONES] TO WR[1]	


```

U 368D, 0869,3C :14886      JSR [CHECK.RESULT]      ;CALL ERROR ROUTINE WITH DELIBERATELY
                  :14887      ;ERRONEOUS DATA TO FLAG SKIP ERROR
U 368E, 8B68,84 :14888      JMP [LOOP.T20.10]      ;LOOP ON ERROR IF ENABLED
                  :14889
T20.11:          :14890      JSR [INC.EN]          ;INCREMENT ERROR NUMBER TO 11
                  :14891      MISC [SET.S1]
U 368F, 8870,5C :14891      MISC [SET.S0]
U 3690, 9060,15 :14892      MISC [CLR.S0]          ;0 CLEAR 1 SET
U 3691, 9050,15 :14893
U 3692, 9030,15 :14894
                  :14895      LOOP.T20.11:
U 3693, DB00,0D :14895      SKIP.IF[STATE.ONE.SET]
U 3694, 5B00,1E :14896      SKIP
U 3695, 8B69,A4 :14897      JMP [T20.12]
U 3696, 2F80,15 :14898      CLR WR[0]
U 3697, 369E,95 :14899      MOV LS[ONES] TO WR[1]
U 3698, 0869,3C :14900      JSR [CHECK.RESULT]      ;CALL ERROR ROUTINE WITH DELIBERATELY
                  :14901      ;ERRONEOUS DATA TO FLAG SKIP ERROR
U 3699, 8B69,34 :14902      JMP [LOOP.T20.11]      ;LOOP ON ERROR IF ENABLED
                  :14903
T20.12:          :14904      JSR [INC.EN]          ;INCREMENT ERROR NUMBER TO 12
U 369A, 8870,5C :14905      LOOP.T20.12:
                  :14906      SKIP.IF[STATE.ZERO.CLR]
U 369B, DB00,0E :14906      SKIP
U 369C, 5B00,1E :14907      JMP [T20.13]
U 369D, 0B6A,24 :14908      CLR WR[0]
U 369E, 2F80,15 :14909      MOV LS[ONES] TO WR[1]
U 369F, 369E,95 :14910      JSR [CHECK.RESULT]      ;CALL ERROR ROUTINE WITH DELIBERATELY
U 36A0, 0869,3C :14911      ;ERRONEOUS DATA TO FLAG SKIP ERROR
                  :14912      ;LOOP ON ERROR IF ENABLED
U 36A1, 0B69,B4 :14913      JMP [LOOP.T20.12]
                  :14914
T20.13:          :14915      JSR [INC.EN]          ;INCREMENT ERROR NUMBER TO 13
U 36A2, 8870,5C :14916      MOV LS[BIT16] TO WR[1]
U 36A3, B660,95 :14917      BIS LS[BIT22] TO WR[1]      ;SET BITS 22 & 16
U 36A4, C76C,95 :14918      MOV WR[1] TO LS[CONTROL.STATUS] ;DISABLE MEMORY REFERENCES
U 36A5, BE80,95 :14919      MISC [SET.CP.ATTN]          ;GET CONSOLES ATTENTION
U 36A6, 10E0,15 :14920      WAIT.T20.13:
                  :14921      JMP [WAIT.T20.13]      ;WAIT FOR CONSOLE TO RESPOND
U 36A7, 0B6A,74 :14922      MISC [SET.S1]
U 36A8, 9060,15 :14923      MISC [SET.S0]
U 36A9, 9050,15 :14924      DECODE.CM.OPC ADRS[0],
                  :14925      JCTL/NO.JUMP.TST      ;BOTH SET
U 36AA, 8402,DE :14926      NOP
U 36AB, DB00,15 :14927      LOOP.T20.13:
                  :14928      SKIP.IF[STATE.ZERO.CLR]
U 36AC, DB00,0E :14929      SKIP
U 36AD, 5B00,1E :14930      JMP [T20.14]
U 36AE, 0B6B,54 :14931      CLR WR[0]
U 36AF, 2F80,15 :14932      MOV LS[ONES] TO WR[1]
U 36B0, 369E,95 :14933      MOV LS[BIT23] TO WR[0]
U 36B1, B66E,15 :14934      MOV WR[0] TO LS[CONTROL.STATUS] ;RESTORE CONTROL&STATUS WORD
U 36B2, 3E80,15 :14935      JSR [CHECK.RESULT]      ;BIT23 = EXPECTED AND RECEIVED DATA N/A
U 36B3, 0869,3C :14936      ;CALL ERROR ROUTINE WITH DELIBERATELY
                  :14937      ;ERRONEOUS DATA TO FLAG SKIP ERROR
U 36B4, 8B6A,C4 :14938      JMP [LOOP.T20.13]      ;LOOP ON ERROR IF ENABLED
                  :14939
T20.14:          :14940      JSR [INC.EN]          ;INCREMENT ERROR NUMBER TO 14
U 36B5, 8870,5C :14940      LOOP.T20.14:
  
```

U 36B6, 5B00,0F	:14941	SKIP.IF[STATE.ONE.CLR]	
U 36B7, 5B00,1E	:14942	SKIP	:ERROR -- SKIP
U 36B8, 8B6B,D4	:14943	JMP [T20.14.B]	
U 36B9, 2F80,15	:14944	CLR WR[0]	
U 36BA, 369E,95	:14945	MOV LS[ONES] TO WR[1]	
U 36BB, 0869,3C	:14946	JSR [CHECK.RESULT]	:CALL ERROR ROUTINE WITH DELIBERATELY
	:14947		:ERRONEOUS DATA TO FLAG SKIP ERROR
U 36BC, 0B6B,64	:14948	JMP [LOOP.T20.14]	:LOOP ON ERROR IF ENABLED
	:14949		
U 36BD, B660,95	:14950	T20.14.B: MOV LS[BIT16] TO WR[1]	:SET BIT 16 (NOTE: OTHERS CLEAR)
U 36BE, BE80,95	:14951	MOV WR[1] TO LS[CONTROL.STATUS]	:ENABLE MEMORY REF.
U 36BF, 10E0,15	:14952	MISC [SET.CP.ATTN]	:GET CONSOLES ATTENTION
	:14953	WAIT.T20.END:	
U 36C0, 8B6C,04	:14954	JMP [WAIT.T20.END]	:WAIT FOR CONSOLE TO RESPOND
	:14955	END.T20:	

	:14956	PAGE	
	:14957	END.SECTION:	
U 36C1, 365C,15	:14958	MOV LS[BIT14] TO WR[0]	;SET UP WR 0 FOR END OF SECTION
U 36C2, 3E80,15	:14959	MOV WR[0] TO LS[CONTROL.STATUS]	;SET UP CONTROL AND STATUS WORD TO
	:14960		; REPORT END OF SECTION
	:14961		
U 36C3, 10E0,15	:14962	MISC [SET.CP.ATTN]	;ISSUE CPU ATTN TO CONSOLE
	:14963	WAIT.EOS:	
U 36C4, 0B6C,44	:14964	JMP [WAIT.EOS]	;LOOP HERE WAITING FOR CONSOLE TO
	:14965		; RESPOND

ZZ-ENKCA-1.0
: ENKCA.MCR
:

Cross Reference Lis 3-MAR-82 N 9
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
Cross Reference Listing - Field Names and Defined Values

Fiche 2 Frame N9
Sequence 323
Rev 01.00
Page 317

ZZ-ENKCA-1.0
: ENKCA.MCR
:

Cross Reference Lis 3-^{B 10}MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
Cross Reference Listing - Macro Names

Fiche 2 Frame B10

Sequence 324

Rev 01.00

Page 318

ZZ-ENKCA-1.0 :
: ENKCA.MCR
:

C 10
MICRO2 1L(03) Cross Reference Lis 3-MAR-82 Fiche 2 Frame C10 Sequence 325
3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module Rev 01.00 Page 319
Cross Reference Listing - Expression Names

ZZ-ENKCA-1.0
; ENKCA.MCR
;

MICRO2 1L(03) Location / Line Num 3-D 10
3-MAR-82 09:34:33 3-MAR-82
Location / Line Number Index

Fiche 2 Frame D10 Sequence 326
ENKCA Micro-diagnostic for DAP module Rev 01.00 Page 320

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
C 0000	1771:	1772:	1773:	1774:	1775:	1776:	1777:	1778:
C 0008	1779:	1780:	1781:	1782:	1783:	1784:	1785:	1786:
C 0010	1790:	1791:	1792:	1793:	1794:	1795:	1796:	1797:
C 0018	1798:	1799:	1800:	1801:	1802:	1803:	1804:	1805:
C 0020	1814:	1815:	1816:	1817:	1818:	1819:	1820:	1821:
C 0028	1822:	1823:	1824:	1825:	1826:	1827:	1828:	1829:
C 0030	1830:	1831:	1832:	1833:	1834:	1835:	1836:	1837:
C 0038	1838:	1839:	1840:	1841:	1842:	1843:	1844:	1845:
C 0040	1854:	1855:	1856:	1857:	1858:	1859:	1860:	1861:
C 0048	1862:	1863:	1864:	1865:	1866:	1867:	1868:	1869:
C 0050	1870:	1871:	1872:	1873:	1874:	1875:	1876:	1877:
C 0058	1878:	1879:	1880:	1881:	1882:	1883:	1884:	1885:
C 0060	1893:	1894:	1895:	1896:	1897:	1898:	1899:	1900:
C 0068	1901:	1902:	1903:	1904:	1905:	1906:	1907:	1908:
C 0070	1909:	1910:	1911:	1912:	1913:	1914:	1915:	1916:
C 0078	1917:	1918:	1919:	1920:	1921:	1922:	1923:	1924:
C 0080	1931:	1933:	1935:	1937:	1940:	1942:	1944:	1946:
C 0088	1949:	1951:	1953:	1955:	1958:	1960:	1962:	1964:
C 0090	1967:	1969:	1971:	1973:	1976:	1978:	1980:	1982:
C 0098	1985:	1987:	1989:	1991:	1994:	1996:	1998:	2000:
C 00A0	2003:	2005:	2007:	2009:	2012:	2014:	2016:	2018:
C 00A8	2021:	2023:	2025:	2027:	2030:	2032:	2034:	2036:
C 00B0	2039:	2041:	2043:	2045:	2048:	2050:	2052:	2054:
C 00B8	2057:	2059:	2061:	2063:	2066:	2068:	2070:	2072:
C 00C0	2080:	2081:	2082:	2083:	2084:	2085:	2086:	2087:
C 00C8	2090:	2091:	2092:	2093:	2094:	2095:	2096:	2097:
C 00D0	2100:	2101:	2102:	2103:	2104:	2105:	2106:	2107:
C 00D8	2110:	2111:	2112:	2113:	2114:	2115:	2116:	2117:
C 00E0	2121:	2122:	2123:	2124:	2125:	2126:	2127:	2128:
C 00E8	2130:	2131:	2132:	2133:	2134:	2135:	2136:	2137:
C 00F0	2141:	2142:	2143:	2144:	2145:	2146:	2147:	2148:
C 00F8	2151:	2152:	2153:	2154:	2155:	2156:	2157:	2158:
C 0100	2166:	2167:	2168:	2169:	2171:	2172:	2173:	2174:
C 0108	2176:	2177:	2178:	2179:	2181:	2182:	2183:	2184:
C 0110	2187:	2188:	2189:	2190:	2192:	2193:	2194:	2195:
C 0118	2197:	2198:	2199:	2200:	2202:	2203:	2204:	2205:
C 0120	2208:	2209:	2210:	2211:	2213:	2214:	2215:	2216:
C 0128	2218:	2219:	2220:	2221:	2223:	2224:	2225:	2226:
C 0130	2229:	2230:	2231:	2232:	2234:	2235:	2236:	2237:
C 0138	2239:	2240:	2241:	2242:	2244:	2245:	2246:	2247:
C 0140	2250:	2251:	2252:	2253:	2255:	2256:	2257:	2258:
C 0148	2260:	2261:	2262:	2263:	2265:	2266:	2267:	2268:
C 0150	2271:	2272:	2273:	2274:	2276:	2277:	2278:	2279:
C 0158	2281:	2282:	2283:	2284:	2286:	2287:	2288:	2289:
C 0160	2292:	2293:	2294:	2295:	2297:	2298:	2299:	2300:
C 0168	2302:	2303:	2304:	2305:	2307:	2308:	2309:	2310:
C 0170	2313:	2314:	2315:	2316:	2318:	2319:	2320:	2321:
C 0178	2323:	2324:	2325:	2326:	2328:	2329:	2330:	2331:
C 0180	2341:	2342:	2343:	2344:	2346:	2347:	2348:	2349:
C 0188	2351:	2352:	2353:	2354:	2356:	2357:	2358:	2359:
C 0190	2362:	2363:	2364:	2365:	2367:	2368:	2369:	2370:
C 0198	2372:	2373:	2374:	2375:	2377:	2378:	2379:	2380:
C 01A0	2383:	2384:	2385:	2386:	2388:	2389:	2390:	2391:

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
C 01A8	2393:	2394:	2395:	2396:	2398:	2399:	2400:	2401:
C 01B0	2404:	2405:	2406:	2407:	2409:	2410:	2411:	2412:
C 01B8	2414:	2415:	2416:	2417:	2419:	2420:	2421:	2422:
C 01C0	2425:	2426:	2427:	2428:	2430:	2431:	2432:	2433:
C 01C8	2435:	2436:	2437:	2438:	2440:	2441:	2442:	2443:
C 01D0	2446:	2447:	2448:	2449:	2451:	2452:	2453:	2454:
C 01D8	2456:	2457:	2458:	2459:	2461:	2462:	2463:	2464:
C 01E0	2467:	2468:	2469:	2470:	2472:	2473:	2474:	2475:
C 01E8	2477:	2478:	2479:	2480:	2482:	2483:	2484:	2485:
C 01F0	2488:	2489:	2490:	2491:	2493:	2494:	2495:	2496:
C 01F8	2498:	2499:	2500:	2501:	2503:	2504:	2505:	2506:

ZZ-ENKCA-1.0
: ENKCA.MCR
:

Location / Line Num 3-
MICRO2 1L(03) 3-MAR-82 09:34:33
Location / Line Number Index

F 10

Fiche 2 Frame F10

Sequence 328
Rev 01.00

Page 322

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 0000	4020:	3873:	3874:	3875:	3876:	3877:	3878:	3879:
U 0008	3880:	3881:	3882:	3883:	3884:	3885:	3886:	3887:
U 0010	3888:	3889:	3890:	3891:	3892:	3893:	3894:	3895:
U 0018	3896:	3897:	3898:	3899:	3900:	3901:	3902:	3903:
U 0020	3904:	3905:	3907:	3909:	3911:	3913:	3915:	3917:
U 0028	3919:	3921:	3923:	3925:	3927:	3929:	3931:	3933:
U 0030	3935:	3937:	3939:	3941:	3943:	3945:	3947:	3949:
U 0038	3951:	3953:	3955:	3957:	3959:	3961:	3963:	3965:
U 0040	3967:	3969:	3971:	3973:	3975:	3977:	3979:	3981:
U 0048	3983:	3985:	3987:	3989:	3991:	3993:	3995:	3997:
U 0050	4029:	4031:	4033:	4036:	4039:	4043:	4045:	4047:
U 0058	4049:	4051:	4053:	4055:	4057:	4059:	4061:	4063:
U 0060	4065:	4067:	4069:	4071:	4073:	4075:	4076:	4077:
U 0068	4078:	4079:	4080:	4081:	4082:	4083:	4084:	4085:
U 0070	4116:	4118:	4120:	4121:	4123:	4124:	4126:	4127:
U 0078	4129:	4130:	4132:	4133:	4135:	4136:	4138:	4139:
U 0080	4141:	4142:	4144:	4145:	4147:	4148:	4150:	4151:
U 0088	4153:	4154:	4156:	4157:	4159:	4160:	4162:	4163:
U 0090	4165:	4166:	4168:	4169:	4171:	4172:	4174:	4175:
U 0098	4177:	4178:	4180:	4181:	4183:	4184:	4186:	4187:
U 00A0	4189:	4190:	4192:	4193:	4195:	4196:	4198:	4199:
U 00A8	4201:	4202:	4204:	4205:	4207:	4208:	4210:	4211:
U 00B0	4213:	4214:	4216:	4217:	4219:	4220:	4222:	4223:
U 00B8	4225:	4226:	4228:	4229:	4231:	4232:	4234:	4235:
U 00C0	4237:	4238:	4240:	4241:	4243:	4244:	4246:	4247:
U 00C8	4249:	4250:	4252:	4253:	4255:	4256:	4258:	4259:
U 00D0	4261:	4262:	4264:	4265:	4267:	4268:	4270:	4271:
U 00D8	4273:	4274:	4276:	4277:	4279:	4280:	4282:	4283:
U 00E0	4285:	4286:	4288:	4289:	4291:	4292:	4294:	4295:
U 00E8	4297:	4298:	4300:	4301:	4303:	4304:	4306:	4307:
U 00F0	4309:	4310:	4312:	4313:	4315:	4316:	4318:	4319:
U 00F8	4321:	4322:	4324:	4325:	4327:	4328:	4330:	4331:
U 0100	4333:	4334:	4336:	4337:	4339:	4340:	4342:	4343:
U 0108	4345:	4346:	4348:	4349:	4351:	4352:	4354:	4355:
U 0110	4357:	4358:	4360:	4361:	4363:	4364:	4366:	4367:
U 0118	4369:	4370:	4372:	4373:	4375:	4376:	4378:	4379:
U 0120	4381:	4382:	4384:	4385:	4387:	4388:	4390:	4391:
U 0128	4393:	4394:	4396:	4397:	4399:	4400:	4402:	4403:
U 0130	4405:	4406:	4408:	4409:	4411:	4412:	4414:	4415:
U 0138	4417:	4418:	4420:	4421:	4423:	4424:	4426:	4427:
U 0140	4429:	4430:	4432:	4433:	4435:	4436:	4438:	4439:
U 0148	4441:	4442:	4444:	4445:	4447:	4448:	4450:	4451:
U 0150	4453:	4454:	4456:	4457:	4459:	4460:	4462:	4463:
U 0158	4465:	4466:	4468:	4469:	4471:	4472:	4474:	4475:
U 0160	4477:	4478:	4487:	4489:	4491:	4492:	4494:	4495:
U 0168	4497:	4498:	4500:	4501:	4503:	4504:	4506:	4507:
U 0170	4509:	4510:	4512:	4513:	4515:	4516:	4518:	4519:
U 0178	4521:	4522:	4524:	4525:	4527:	4528:	4530:	4531:
U 0180	4533:	4534:	4536:	4537:	4539:	4540:	4542:	4543:
U 0188	4545:	4546:	4548:	4549:	4551:	4552:	4554:	4555:
U 0190	4557:	4558:	4560:	4561:	4563:	4564:	4566:	4567:
U 0198	4569:	4570:	4572:	4573:	4575:	4576:	4578:	4579:
U 01A0	4581:	4582:	4584:	4585:	4587:	4588:	4590:	4591:

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 01A8	4593:	4594:	4596:	4597:	4599:	4600:	4602:	4603:
U 01B0	4605:	4606:	4608:	4609:	4611:	4612:	4614:	4615:
U 01B8	4617:	4618:	4620:	4621:	4623:	4624:	4626:	4627:
U 01C0	4629:	4630:	4632:	4633:	4635:	4636:	4638:	4639:
U 01C8	4641:	4642:	4644:	4645:	4647:	4648:	4650:	4651:
U 01D0	4653:	4654:	4656:	4657:	4659:	4660:	4662:	4663:
U 01D8	4665:	4666:	4668:	4669:	4671:	4672:	4674:	4675:
U 01E0	4677:	4678:	4680:	4681:	4683:	4684:	4686:	4687:
U 01E8	4689:	4690:	4692:	4693:	4695:	4696:	4698:	4699:
U 01F0	4701:	4702:	4704:	4705:	4707:	4708:	4710:	4711:
U 01F8	4713:	4714:	4716:	4717:	4719:	4720:	4722:	4723:
U 0200	4725:	4726:	4728:	4729:	4731:	4732:	4734:	4735:
U 0208	4737:	4738:	4740:	4741:	4743:	4744:	4746:	4747:
U 0210	4749:	4750:	4752:	4753:	4755:	4756:	4758:	4759:
U 0218	4761:	4762:	4764:	4765:	4767:	4768:	4770:	4771:
U 0220	4773:	4774:	4776:	4777:	4779:	4780:	4782:	4783:
U 0228	4785:	4786:	4788:	4789:	4791:	4792:	4794:	4795:
U 0230	4797:	4798:	4800:	4801:	4803:	4804:	4806:	4807:
U 0238	4809:	4810:	4812:	4813:	4815:	4816:	4818:	4819:
U 0240	4821:	4822:	4824:	4825:	4827:	4828:	4830:	4831:
U 0248	4833:	4834:	4836:	4837:	4839:	4840:	4842:	4843:
U 0250	4845:	4846:	4848:	4849:				
U 0258 - 05FF	Unused							
U 0600	4876:	4877:	4879:	4882:	4883:	4884:	4898:	4899:
U 0608	4960:	4962:	4964:	4966:	4967:	5022:	5023:	5025:
U 0610	5027:	5029:	5032:	5034:	5035:	5036:	5040:	5041:
U 0618	5042:	5046:	5047:	5048:	5050:	5053:	5054:	5109:
U 0620	5111:	5112:	5115:	5117:	5121:	5123:	5127:	5133:
U 0628	5135:	5136:	5142:	5144:	5146:	5147:	5148:	5154:
U 0630	5155:	5156:	5157:	5158:	5159:	5160:	5163:	5165:
U 0638	5166:	5170:	5220:	5222:	5224:	5226:	5229:	5230:
U 0640	5232:	5233:	5288:	5290:	5291:	5294:	5296:	5300:
U 0648	5302:	5306:	5312:	5314:	5315:	5321:	5322:	5323:
U 0650	5324:	5325:	5326:	5329:	5379:	5381:	5384:	5386:
U 0658	5389:	5390:	5392:	5393:	5441:	5446:	5448:	5450:
U 0660	5451:	5452:	5502:	5504:	5507:	5511:	5513:	5514:
U 0668	5553:	5554:	5555:	5558:	5559:	5560:	5563:	5564:
U 0670	5565:	5568:	5569:	5570:	5573:	5574:	5578:	5579:
U 0678	5580:	5619:	5620:	5621:	5625:	5626:	5627:	5631:
U 0680	5632:	5633:	5637:	5638:	5642:	5643:	5647:	5650:
U 0688	5652:	5699:	5700:	5701:	5702:	5703:	5750:	5751:
U 0690	5752:	5753:	5754:	5822:	5824:	5827:	5828:	5830:
U 0698	5832:	5833:	5834:	5836:	5840:	5842:	5844:	5848:
U 06A0	5849:	5852:	5853:	5855:	5857:	5858:	5860:	5862:
U 06A8	5867:	5868:	5870:	5872:	5873:	5879:	5880:	5883:
U 06B0	5884:	5885:	5890:	5891:	5893:	5894:	5896:	5898:
U 06B8	5899:	5901:	5903:	5905:	5907:	5909:	5916:	5917:
U 06C0	5919:	5922:	5935:	5936:	5938:	5939:	5940:	5941:
U 06C8	5942:	5943:	5944:	5945:	5946:	5950:	5953:	5954:
U 06D0	5955:	5956:	5957:	5958:	5959:	5960:	5961:	5962:
U 06D8	5963:	5964:	5965:	5966:	5967:	5968:	5971:	5974:
U 06E0	5977:	5979:	5980:	5981:	5982:	5985:	5988:	5989:
U 06E8	5991:	5992:	5994:	5996:	5997:			

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 06F0 - 06FF Unused								
U 0700	6005:	6006:	6007:	6008:	6009:	6011:	6012:	6013:
U 0708	6014:	6016:	6017:	6018:	6020:	6021:	6022:	6023:
U 0710	6024:	6026:	6027:	6028:	6029:	6030:	6032:	6034:
U 0718	6035:	6036:	6037:	6130:	6131:	6133:	6135:	6136:
U 0720	6137:	6139:	6140:	6141:	6142:	6143:	6144:	6145:
U 0728	6146:	6154:						
U 0730 - 073F Unused								
U 0740	6285:	6287:	6288:	6289:	6291:	6293:	6294:	6295:
U 0748	6297:	6299:	6300:	6301:	6303:	6305:	6306:	6307:
U 0750	6309:	6311:	6312:	6313:	6315:	6317:	6318:	6319:
U 0758	6321:	6323:	6324:	6325:	6327:	6329:	6330:	6331:
U 0760	6333:	6335:	6336:	6337:	6339:	6341:	6342:	6343:
U 0768	6345:	6347:	6348:	6453:	6455:	6458:	6460:	6462:
U 0770	6463:	6465:	6466:	6467:	6468:	6469:	6470:	6475:
U 0778	6476:	6478:	6479:	6480:	6482:	6483:	6486:	6488:
U 0780	6489:	6490:	6491:	6493:	6496:	6502:	6504:	6505:
U 0788	6509:	6510:	6512:	6515:	6517:	6518:	6522:	6524:
U 0790	6532:	6533:	6536:	6537:	6539:	6540:	6541:	6543:
U 0798	6545:	6546:	6549:	6550:	6553:	6555:	6556:	6557:
U 07A0	6558:	6560:	6561:	6564:	6565:	6567:	6568:	6570:
U 07A8	6572:	6573:	6576:	6577:	6580:	6582:	6583:	6584:
U 07B0	6587:	6588:	6591:	6592:	6594:	6595:	6599:	6600:
U 07B8	6603:	6605:	6606:	6607:	6609:	6611:	6612:	6613:
U 07C0	6616:	6619:	6625:	6627:	6628:	6629:	6632:	6634:
U 07C8	6635:	6636:	6639:	6641:	6642:	6643:	6646:	6648:
U 07D0	6649:	6650:	6780:	6781:	6783:	6785:	6786:	6787:
U 07D8	6789:	6790:	6791:	6792:	6793:	6794:	6795:	6796:
U 07E0	6804:	6805:	6806:	6808:	6810:	6812:	6814:	6815:
U 07E8	6816:	6818:	6819:	6821:	6822:	6824:	6825:	6826:
U 07F0	6827:	6828:	6830:	6831:	6833:	6834:	6835:	6836:
U 07F8	6837:						6169:	6170:
U 0800	6172:	6173:	6175:	6176:	6177:	6178:		
U 0808					6205:	6206:	6208:	6209:
U 0810	6211:	6212:	6213:	6214:				
U 0818					6217:	6218:	6220:	6221:
U 0820	6223:	6224:	6225:	6226:				
U 0828 - 0837 Unused								
U 0838					6229:	6230:	6232:	6233:
U 0840	6235:	6236:	6237:	6238:				
U 0848 - 0877 Unused								
U 0878					6241:	6242:	6244:	6245:
U 0880	6247:	6248:	6249:	6250:				
U 0888 - 08F7 Unused								
U 08F8					6253:	6254:	6256:	6257:
U 0900	6259:	6260:	6261:	6262:				
U 0908 - 09F7 Unused								
U 09F8					6265:	6266:	6268:	6269:
U 0A00	6271:	6272:	6273:	6274:				
U 0A08 - 0BF7 Unused								
U 0BF8					6277:	6278:	6280:	6281:
U 0C00	6282:							
U 0C08 - 0FFF Unused								

ZZ-ENKCA-1.0
: ENKCA.MCR
:

MICRO2 1L(03) Location / Line Num 3-I 10
3-MAR-82 09:34:33
Location / Line Number Index

Fiche 2 Frame I10 Sequence 331
ENKCA Micro-diagnostic for DAP module Rev 01.00 Page 325

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1000					6193:	6194:	6196:	6197:
U 1008	6199:	6200:	6201:	6202:				
U 1010	7098:	7099:	7100:	7101:	7102:	7103:	7104:	7106:
U 1018	7107:	7108:	7110:	7111:	7112:	7114:	7116:	7117:
U 1020	7118:	7119:	7121:	7122:	7123:	7125:	7126:	7128:
U 1028	7129:	7130:	7132:	7133:	7134:	7136:	7137:	7138:
U 1030	7140:	7141:	7142:	7144:	7145:	7146:	7148:	7150:
U 1038	7151:	7152:	7154:	7155:	7156:	7158:	7159:	7160:
U 1040	7162:	7163:	7164:	7166:	7167:	7168:	7170:	7171:
U 1048	7172:	7174:	7175:	7176:	7178:	7179:	7180:	7182:
U 1050	7183:	7184:	7186:	7187:	7188:	7190:	7191:	7192:
U 1058	7194:	7195:	7196:	7198:	7199:	7201:	7203:	7204:
U 1060	7205:	7207:	7208:	7209:	7211:	7212:	7213:	7215:
U 1068	7216:	7217:	7219:	7220:	7221:	7223:	7224:	7225:
U 1070	7227:	7228:	7229:	7231:	7232:	7233:	7235:	7236:
U 1078	7237:	7239:	7240:	7241:	7243:	7244:	7245:	7247:
U 1080	7248:	7249:	7251:	7252:	7254:	7256:	7257:	7258:
U 1088	7260:	7261:						
U 1090 - 15FF Unused								
U 1600	6181:	6182:	6184:	6185:	6187:	6188:	6189:	6190:
U 1608 - 17F7 Unused								
U 17F8						6157:	6158:	6160:
U 1800	6161:	6163:	6164:	6165:	6166:			
U 1808 - 180F Unused								
U 1810	7367:	7368:	7370:	7372:	7373:	7374:	7375:	7376:
U 1818	7379:	7380:	7381:	7382:	7383:	7384:	7385:	7386:
U 1820	7387:	7388:	7389:	7390:	7391:	7393:	7394:	7396:
U 1828	7397:	7398:	7399:	7400:	7402:	7403:	7404:	7406:
U 1830	7407:	7409:	7410:	7411:	7412:	7413:	7414:	7416:
U 1838	7417:	7419:	7420:	7421:	7422:	7423:	7425:	7426:
U 1840	7427:	7429:	7430:	7432:	7433:	7435:	7436:	7438:
U 1848	7439:	7440:	7442:	7443:	7444:	7445:	7447:	7448:
U 1850	7449:	7451:	7452:	7455:	7456:	7457:	7458:	7460:
U 1858	7461:	7462:	7463:	7464:	7466:	7467:	7468:	7469:
U 1860	7470:	7472:	7473:	7474:	7475:	7476:	7478:	7479:
U 1868	7481:	7482:	7483:	7485:	7486:	7487:	7488:	7490:
U 1870	7491:	7492:	7494:	7495:	7498:	7499:	7500:	7501:
U 1878	7503:	7504:	7505:	7506:	7507:	7509:	7510:	7511:
U 1880	7512:	7513:	7515:	7516:	7517:	7518:	7522:	7523:
U 1888	7524:	7525:	7526:	7528:	7529:	7530:	7531:	7532:
U 1890	7533:	7534:	7537:	7538:	7540:	7541:	7542:	7544:
U 1898	7545:	7546:	7547:	7549:	7550:	7551:	7553:	7554:
U 18A0	7556:	7557:	7558:	7559:	7561:	7562:	7563:	7564:
U 18A8	7566:	7567:	7568:	7570:	7571:	7573:	7574:	7575:
U 18B0	7576:	7578:	7579:	7580:	7581:	7582:	7583:	7584:
U 18B8	7585:	7586:	7587:	7589:	7590:	7592:	7593:	7594:
U 18C0	7596:	7597:	7598:	7599:	7601:	7602:	7603:	7605:
U 18C8	7606:	7608:	7609:	7610:	7611:	7613:	7614:	7615:
U 18D0	7616:	7618:	7619:	7620:	7622:	7623:	7625:	7626:
U 18D8	7627:	7628:	7630:	7631:	7632:	7633:	7634:	7728:
U 18E0	7729:	7731:	7733:	7734:	7735:	7736:	7737:	7738:
U 18E8	7739:	7740:	7741:	7742:	7743:	7744:	7749:	7750:
U 18F0	7752:	7753:	7754:	7755:	7756:	7757:	7759:	7760:

ZZ-ENKCA-1.0
: ENKCA.MCR
:

MICRO2 1L(03) Location / Line Num 3-MAR-82
Location / Line Number Index

J 10

Fiche 2 Frame J10

Sequence 332

ENKCA Micro-diagnostic for DAP module Rev 01.00 Page 326

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 18F8	7761:	7763:	7765:	7766:	7769:	7770:	7771:	7772:
U 1900	7774:	7776:	7777:	7778:	7779:	7780:	7782:	7783:
U 1908	7784:	7786:	7788:	7789:	7791:	7792:	7794:	7795:
U 1910	7796:	7797:	7798:	7799:	7801:	7802:	7803:	7805:
U 1918	7807:	7808:	7810:	7812:	7813:	7814:	7816:	7817:
U 1920	7818:	7819:	7820:	7821:	7823:	7824:	7825:	7827:
U 1928	7829:	7830:	7832:	7834:	7835:	7836:	7838:	7839:
U 1930	7840:	7841:	7842:	7843:	7845:	7846:	7847:	7849:
U 1938	7851:	7852:	7854:	7855:	7856:	7857:	7858:	7860:
U 1940	7861:	7863:	7864:	7865:	7867:	7868:	7869:	7870:
U 1948	7871:	7872:	7874:	7875:	7876:	7878:	7880:	7881:
U 1950	7883:	7884:	7885:	7886:	7888:	7889:	7890:	7891:
U 1958	7892:	7894:	7896:	7897:	7898:	7899:	7900:	7902:
U 1960	7903:	7904:	7905:	7906:	7908:	7909:	7910:	7912:
U 1968	7914:	7915:	7917:	7919:	7920:	7921:	7922:	7923:
U 1970	7924:	7925:	7927:	7928:	7929:	7930:	7931:	7933:
U 1978	7934:	7935:	7937:	7939:	7940:	7942:	7943:	8042:
U 1980	8043:	8045:	8047:	8048:	8049:	8050:	8051:	8052:
U 1988	8053:	8054:	8055:	8056:	8063:	8064:	8066:	8067:
U 1990	8068:	8069:	8070:	8071:	8073:	8074:	8075:	8077:
U 1998	8079:	8080:	8082:	8084:	8085:	8087:	8088:	8089:
U 19A0	8090:	8091:	8092:	8094:	8095:	8096:	8098:	8100:
U 19A8	8101:	8103:	8104:	8105:	8107:	8108:	8109:	8110:
U 19B0	8111:	8112:	8114:	8115:	8116:	8118:	8120:	8121:
U 19B8	8123:	8125:	8126:	8128:	8129:	8130:	8131:	8132:
U 19C0	8133:	8135:	8136:	8137:	8139:	8141:	8142:	8144:
U 19C8	8146:	8147:	8149:	8150:	8151:	8152:	8153:	8154:
U 19D0	8156:	8157:	8158:	8160:	8162:	8163:	8165:	8166:
U 19D8	8167:	8168:	8169:	8171:	8172:	8174:	8175:	8176:
U 19E0	8178:	8179:	8180:	8181:	8182:	8183:	8185:	8186:
U 19E8	8187:	8189:	8191:	8192:	8194:	8196:	8197:	8198:
U 19F0	8199:	8200:	8201:	8202:	8203:	8205:	8207:	8208:
U 19F8	8210:	8211:	8212:	8213:	8214:	8215:	8217:	8218:
U 1A00	8219:	8221:	8223:	8224:	8226:	8228:	8229:	8230:
U 1A08	8231:	8233:	8234:	8235:	8236:	8237:	8238:	8240:
U 1A10	8241:	8242:	8244:	8246:	8247:	8249:	8250:	8347:
U 1A18	8348:	8350:	8352:	8353:	8354:	8355:	8356:	8357:
U 1A20	8358:	8359:	8360:	8361:	8369:	8370:	8372:	8373:
U 1A28	8374:	8375:	8376:	8378:	8379:	8380:	8382:	8384:
U 1A30	8385:	8387:	8389:	8390:	8392:	8393:	8394:	8395:
U 1A38	8396:	8398:	8399:	8400:	8402:	8404:	8405:	8407:
U 1A40	8408:	8409:	8411:	8412:	8413:	8414:	8415:	8417:
U 1A48	8418:	8419:	8421:	8423:	8424:	8426:	8428:	8429:
U 1A50	8431:	8432:	8433:	8434:	8435:	8437:	8438:	8439:
U 1A58	8441:	8443:	8444:	8446:	8448:	8449:	8451:	8452:
U 1A60	8453:	8454:	8455:	8457:	8458:	8459:	8461:	8463:
U 1A68	8464:	8466:	8467:	8468:	8470:	8471:	8473:	8475:
U 1A70	8476:	8477:	8479:	8480:	8481:	8482:	8484:	8485:
U 1A78	8486:	8488:	8490:	8491:	8493:	8494:	8495:	8497:
U 1A80	8498:	8499:	8500:	8502:	8504:	8505:	8506:	8507:
U 1A88	8509:	8510:	8511:	8512:	8513:	8515:	8516:	8517:
U 1A90	8519:	8521:	8522:	8524:	8526:	8527:	8528:	8529:
U 1A98	8530:	8531:	8533:	8534:	8535:	8536:	8537:	8539:

ZZ-ENKCA-1.0
: ENKCA.MCR
:

MICRO2 1L(03) Location / Line Num 3-^{K 10}MAR-82
Location / Line Number Index 09:34:33

Fiche 2 Frame K10
ENKCA Micro-diagnostic for DAP module

Sequence 333
Rev 01.00 Page 327

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1AA0	8540:	8541:	8543:	8545:	8546:	8548:	8549:	8550:
U 1AA8	8552:	8553:	8554:	8630:	8631:	8633:	8635:	8636:
U 1AB0	8637:	8638:	8639:	8640:	8641:	8642:	8643:	8644:
U 1AB8	8650:	8651:	8652:	8653:	8655:	8656:	8657:	8658:
U 1AC0	8659:	8660:	8661:	8662:	8664:	8665:	8666:	8667:
U 1AC8	8668:	8669:	8670:	8672:	8673:	8674:	8676:	8678:
U 1AD0	8679:	8681:	8682:	8683:	8685:	8686:	8687:	8688:
U 1AD8	8689:	8690:	8692:	8693:	8694:	8696:	8698:	8699:
U 1AE0	8701:	8703:	8704:	8705:	8706:	8707:	8708:	8709:
U 1AE8	8710:	8711:	8713:	8714:	8715:	8716:	8717:	8718:
U 1AF0	8719:	8721:	8722:	8723:	8725:	8727:	8728:	8730:
U 1AF8	8731:	8732:	8734:	8735:	8736:	8737:	8738:	8739:
U 1B00	8741:	8742:	8743:	8745:	8747:	8748:	8750:	8752:
U 1B08	8753:	8754:	8755:	8756:	8757:	8758:	8759:	8760:
U 1B10	8762:	8763:	8764:	8765:	8766:	8767:	8768:	8770:
U 1B18	8771:	8772:	8774:	8776:	8777:	8779:	8780:	8781:
U 1B20	8782:	8783:	8785:	8786:	8787:	8788:	8789:	8790:
U 1B28	8792:	8793:	8794:	8796:	8798:	8799:	8801:	8803:
U 1B30	8804:	8805:	8806:	8807:	8808:	8809:	8810:	8811:
U 1B38	8813:	8814:	8815:	8816:	8817:	8818:	8819:	8821:
U 1B40	8822:	8823:	8825:	8827:	8828:	8830:	8831:	8832:
U 1B48	8834:	8835:	8836:	8837:	8838:	8839:	8841:	8842:
U 1B50	8843:	8845:	8847:	8848:	8850:	8851:	8852:	8853:
U 1B58	8940:	8941:	8943:	8945:	8946:	8947:	8948:	8949:
U 1B60	8950:	8951:	8952:	8954:	8955:	8956:	8957:	8958:
U 1B68	8959:	8961:	8962:	8963:	8964:	8965:	8966:	8967:
U 1B70	8969:	8970:	8971:	8972:	8974:	8975:	8976:	8977:
U 1B78	8978:	8979:	8981:	8982:	8983:	8985:	8986:	8987:
U 1B80	8988:	8989:	8990:	8991:	8993:	8994:	8995:	8996:
U 1B88	8997:	8998:	8999:	9000:	9002:	9003:	9004:	9005:
U 1B90	9006:	9008:	9009:	9010:	9011:	9012:	9013:	9015:
U 1B98	9016:	9111:	9112:	9114:	9116:	9117:	9118:	9119:
U 1BA0	9120:	9121:	9122:	9123:	9124:	9125:	9129:	9131:
U 1BA8	9132:	9134:	9135:	9136:	9137:	9138:	9139:	9141:
U 1BB0	9142:	9143:	9145:	9147:	9148:	9150:	9151:	9153:
U 1BB8	9154:	9155:	9156:	9157:	9158:	9160:	9161:	9162:
U 1BC0	9164:	9166:	9167:	9169:	9170:	9172:	9173:	9175:
U 1BC8	9176:	9177:	9178:	9179:	9180:	9182:	9183:	9184:
U 1BD0	9186:	9188:	9189:	9191:	9192:	9193:	9195:	9196:
U 1BD8	9197:	9198:	9200:	9202:	9203:	9205:	9206:	9207:
U 1BE0	9208:	9210:	9211:	9212:	9213:	9214:	9216:	9217:
U 1BE8	9218:	9220:	9222:	9223:	9225:	9226:	9227:	9229:
U 1BF0	9230:	9231:	9232:	9234:	9235:	9237:	9238:	9240:
U 1BF8	9241:	9242:	9243:	9244:	9245:	9246:	9248:	9249:
U 1C00	9250:	9252:	9254:	9255:	9257:	9258:	9260:	9261:
U 1C08	9262:	9263:	9264:	9265:	9266:	9268:	9269:	9270:
U 1C10	9272:	9274:	9275:	9277:	9278:	9280:	9281:	9283:
U 1C18	9284:	9285:	9286:	9287:	9288:	9289:	9291:	9292:
U 1C20	9293:	9295:	9297:	9298:	9300:	9301:	9302:	9304:
U 1C28	9305:	9306:	9307:	9309:	9311:	9312:	9314:	9315:
U 1C30	9316:	9317:	9319:	9320:	9321:	9322:	9323:	9325:
U 1C38	9326:	9327:	9329:	9331:	9332:	9334:	9335:	9336:
U 1C40	9337:	9339:	9340:	9341:	9342:	9344:	9345:	9347:

ZZ-ENKCA-1.0
: ENKCA.MCR
:

M 10
Location / Line Num 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33 ENKCA Micro-diagnostic for DAP module
Location / Line Number Index
Fiche 2 Frame M10
Sequence 335
Rev 01.00 Page 329

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1DF0	10164:	10165:	10166:	10168:	10169:	10170:	10171:	10172:
U 1DF8	10173:	10175:	10176:	10177:	10178:	10179:	10180:	10181:
U 1E00	10183:	10184:	10185:	10186:	10187:	10188:	10189:	10191:
U 1E08	10192:	10193:	10194:	10195:	10196:	10198:	10199:	10200:
U 1E10	10201:	10202:	10203:	10205:	10206:	10207:	10208:	10209:
U 1E18	10210:	10211:	10213:	10214:	10215:	10216:	10217:	10218:
U 1E20	10219:	10220:	10221:	10223:	10224:	10225:	10346:	10347:
U 1E28	10349:	10351:	10352:	10353:	10354:	10355:	10356:	10357:
U 1E30	10358:	10360:	10361:	10362:	10363:	10364:	10365:	10366:
U 1E38	10367:	10369:	10370:	10371:	10372:	10373:	10374:	10375:
U 1E40	10377:	10378:	10379:	10380:	10381:	10382:	10383:	10384:
U 1E48	10386:	10387:	10388:	10389:	10390:	10391:	10392:	10393:
U 1E50	10395:	10396:	10397:	10398:	10399:	10400:	10401:	10402:
U 1E58	10403:	10404:	10406:	10407:	10408:	10409:	10410:	10411:
U 1E60	10412:	10413:	10415:	10416:	10417:	10418:	10419:	10420:
U 1E68	10421:	10422:	10424:	10425:	10426:	10427:	10428:	10429:
U 1E70	10430:	10431:	10433:	10434:	10435:	10436:	10437:	10438:
U 1E78	10439:	10440:	10441:	10442:	10443:	10445:	10446:	10447:
U 1E80	10526:	10527:	10529:	10531:	10532:	10533:	10534:	10535:
U 1E88	10536:	10537:	10538:	10540:	10541:	10542:	10543:	10544:
U 1E90	10545:	10546:	10547:	10549:	10550:	10551:	10552:	10553:
U 1E98	10554:	10555:	10557:	10558:	10559:	10560:	10561:	10562:
U 1EA0	10563:	10564:	10565:	10566:	10568:	10569:	10570:	10571:
U 1EA8	10573:	10574:	10575:	10576:	10577:	10578:	10579:	10580:
U 1EB0	10581:	10582:	10583:	10584:	10586:	10587:	10588:	10644:
U 1EB8	10645:	10647:	10649:	10650:	10651:	10652:	10653:	10654:
U 1EC0	10655:	10656:	10658:	10659:	10660:	10661:	10662:	10663:
U 1EC8	10664:	10665:	10667:	10668:	10669:	10670:	10671:	10672:
U 1ED0	10728:	10729:	10731:	10733:	10734:	10735:	10736:	10737:
U 1ED8	10738:	10739:	10740:	10742:	10743:	10744:	10745:	10746:
U 1EE0	10747:	10748:	10749:	10750:	10752:	10753:	10754:	10755:
U 1EE8	10756:	10757:	10758:	10804:	10805:	10807:	10809:	10810:
U 1EF0	10811:	10812:	10813:	10814:	10815:	10816:	10818:	10819:
U 1EF8	10820:	10821:	10822:	10823:	10874:	10875:	10877:	10879:
U 1F00	10880:	10881:	10882:	10883:	10884:	10885:	10886:	10888:
U 1F08	10889:	10890:	10891:	10892:	10893:	10894:	10895:	10896:
U 1F10	10897:	10898:	10899:	11012:	11013:	11015:	11017:	11018:
U 1F18	11019:	11020:	11021:	11022:	11023:	11024:	11025:	11026:
U 1F20	11027:	11028:	11030:	11031:	11032:	11033:	11034:	11035:
U 1F28	11036:	11037:	11038:	11039:	11040:	11041:	11042:	11043:
U 1F30	11044:	11046:	11047:	11048:	11049:	11050:	11051:	11053:
U 1F38	11054:	11055:	11056:	11058:	11059:	11060:	11061:	11062:
U 1F40	11063:	11064:	11065:	11066:	11067:	11068:	11069:	11070:
U 1F48	11071:	11072:	11074:	11075:	11076:	11077:	11078:	11079:
U 1F50	11081:	11082:	11083:	11085:	11086:	11087:	11088:	11089:
U 1F58	11090:	11091:	11092:	11093:	11094:	11095:	11096:	11097:
U 1F60	11098:	11099:	11100:	11101:	11103:	11104:	11105:	11106:
U 1F68	11107:	11108:	11110:	11111:	11112:	11113:	11114:	11116:
U 1F70	11117:	11118:	11119:	11120:	11121:	11122:	11123:	11124:
U 1F78	11125:	11126:	11127:	11128:	11129:	11130:	11131:	11133:
U 1F80	11134:	11135:	11136:	11137:	11138:	11140:	11141:	11142:
U 1F88 - 1FF7 Unused								
U 1FF8								

6840:

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 2000	6842:	6843:	6845:	6846:	6847:	6848:		
U 2008								6987:
U 2010	6988:	6990:	6991:	6992:	6994:	6995:	6996:	6998:
U 2018	6999:	7000:	7001:					6970:
U 2020	6971:	6973:	6974:	6975:	6977:	6978:	6979:	6981:
U 2028	6982:	6983:	6984:					
U 2030 - 2037	Unused							
U 2038								6953:
U 2040	6954:	6956:	6957:	6958:	6960:	6961:	6962:	6964:
U 2048	6965:	6966:	6967:					
U 2050 - 2077	Unused							
U 2078								6936:
U 2080	6937:	6939:	6940:	6941:	6943:	6944:	6945:	6947:
U 2088	6948:	6949:	6950:					
U 2090 - 20F7	Unused							
U 20F8								6919:
U 2100	6920:	6922:	6923:	6924:	6926:	6927:	6928:	6930:
U 2108	6931:	6932:	6933:					
U 2110 - 21F7	Unused							
U 21F8								6902:
U 2200	6903:	6905:	6906:	6907:	6909:	6910:	6911:	6913:
U 2208	6914:	6915:	6916:					
U 2210 - 22FF	Unused							
U 2300	7091:							7004:
U 2308	7005:	7007:	7008:	7009:	7011:	7012:	7013:	7015:
U 2310	7016:	7017:	7018:					
U 2318 - 23F7	Unused							
U 23F8								6885:
U 2400	6886:	6888:	6889:	6890:	6892:	6893:	6894:	6896:
U 2408	6897:	6898:	6899:					
U 2410 - 24FF	Unused							
U 2500	7093:			7021:	7022:	7024:	7025:	7026:
U 2508	7028:	7029:	7030:	7032:	7033:	7034:	7035:	
U 2510 - 25FF	Unused							
U 2600	7095:	7038:	7039:	7041:	7042:	7043:	7045:	7046:
U 2608	7047:	7049:	7050:	7051:	7052:	7053:		
U 2610 - 26FF	Unused							
U 2700	7056:	7057:	7059:	7060:	7061:	7063:	7064:	7065:
U 2708	7067:	7068:	7069:	7071:	7072:	7073:	7074:	7076:
U 2710	7077:	7078:	7080:	7081:	7082:	7084:	7085:	7087:
U 2718	7088:	7089:						
U 2720 - 27F7	Unused							
U 27F8								6868:
U 2800	6869:	6871:	6872:	6873:	6875:	6876:	6877:	6879:
U 2808	6880:	6881:	6882:					
U 2810 - 2FF7	Unused							
U 2FF8								6851:
U 3000	6852:	6854:	6855:	6856:	6858:	6859:	6860:	6862:
U 3008	6863:	6864:	6865:					
U 3010 - 301F	Unused							
U 3020	11298:	11299:	11301:	11303:	11304:	11305:	11306:	11307:
U 3028	11308:	11309:	11310:	11311:	11312:	11313:	11314:	11315:
U 3030	11316:	11317:	11320:	11322:	11323:	11325:	11326:	11327:

ZZ-ENKCA-1.0
: ENKCA.MCR
:

Location / Line Num 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
Location / Line Number Index

Fiche 2 Frame B11
ENKCA Micro-diagnostic for DAP module

Sequence 337
Rev 01.00 Page 331

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 3038	11328:	11329:	11331:	11332:	11334:	11335:	11336:	11337:
U 3040	11338:	11340:	11341:	11343:	11344:	11345:	11346:	11349:
U 3048	11350:	11351:	11352:	11354:	11355:	11356:	11357:	11359:
U 3050	11360:	11361:	11362:	11364:	11365:	11366:	11367:	11368:
U 3058	11370:	11371:	11373:	11374:	11375:	11376:	11377:	11378:
U 3060	11380:	11381:	11383:	11384:	11385:	11387:	11388:	11389:
U 3068	11391:	11392:	11393:	11394:	11395:	11397:	11398:	11399:
U 3070	11401:	11402:	11403:	11404:	11405:	11407:	11408:	11410:
U 3078	11411:	11412:	11413:	11416:	11417:	11418:	11419:	11421:
U 3080	11422:	11423:	11424:	11426:	11427:	11428:	11429:	11431:
U 3088	11432:	11433:	11434:	11435:	11437:	11438:	11440:	11441:
U 3090	11442:	11443:	11444:	11445:	11446:	11448:	11449:	11451:
U 3098	11452:	11453:	11455:	11456:	11457:	11459:	11460:	11461:
U 30A0	11462:	11463:	11465:	11466:	11468:	11469:	11470:	11472:
U 30A8	11473:	11475:	11476:	11477:	11478:	11479:	11482:	11483:
U 30B0	11484:	11485:	11487:	11488:	11489:	11490:	11492:	11493:
U 30B8	11494:	11495:	11497:	11498:	11499:	11500:	11501:	11503:
U 30C0	11504:	11506:	11507:	11508:	11509:	11510:	11512:	11513:
U 30C8	11515:	11516:	11517:	11518:	11520:	11521:	11522:	11523:
U 30D0	11525:	11526:	11527:	11528:	11529:	11531:	11532:	11533:
U 30D8	11535:	11536:	11537:	11539:	11540:	11542:	11543:	11544:
U 30E0	11545:	11546:	11549:	11550:	11551:	11552:	11554:	11555:
U 30E8	11556:	11557:	11559:	11560:	11561:	11562:	11564:	11565:
U 30F0	11566:	11567:	11568:	11570:	11571:	11573:	11574:	11575:
U 30F8	11576:	11577:	11579:	11580:	11680:	11681:	11683:	11685:
U 3100	11686:	11687:	11688:	11689:	11690:	11691:	11692:	11693:
U 3108	11694:	11695:	11697:	11698:	11699:	11700:	11701:	11702:
U 3110	11703:	11705:	11706:	11707:	11708:	11709:	11710:	11711:
U 3118	11713:	11714:	11715:	11716:	11717:	11718:	11720:	11721:
U 3120	11722:	11723:	11724:	11725:	11726:	11728:	11729:	11730:
U 3128	11731:	11732:	11733:	11735:	11736:	11737:	11738:	11739:
U 3130	11932:	11933:	11935:	11937:	11938:	11939:	11940:	11941:
U 3138	11942:	11943:	11944:	11945:	11946:	11947:	11951:	11952:
U 3140	11953:	11954:	11955:	11956:	11959:	11960:	11961:	11962:
U 3148	11963:	11964:	11967:	11968:	11969:	11970:	11971:	11972:
U 3150	11973:	11974:	11975:	11978:	11979:	11980:	11981:	11982:
U 3158	11983:	11986:	11987:	11988:	11989:	11990:	11991:	11994:
U 3160	11995:	11996:	11997:	11998:	11999:	12000:	12001:	12002:
U 3168	12003:	12006:	12007:	12008:	12009:	12010:	12011:	12014:
U 3170	12015:	12016:	12017:	12018:	12019:	12022:	12023:	12024:
U 3178	12025:	12026:	12027:	12028:	12029:	12032:	12033:	12034:
U 3180	12035:	12036:	12037:	12040:	12041:	12042:	12043:	12044:
U 3188	12045:	12048:	12049:	12050:	12051:	12052:	12053:	12054:
U 3190	12055:	12056:	12057:	12058:	12060:	12062:	12063:	12064:
U 3198	12065:	12066:	12067:	12068:	12070:	12072:	12073:	12075:
U 31A0	12076:	12077:	12079:	12080:	12081:	12082:	12083:	12085:
U 31A8	12086:	12087:	12088:	12090:	12091:	12092:	12093:	12094:
U 31B0	12096:	12098:	12099:	12101:	12102:	12103:	12105:	12106:
U 31B8	12107:	12108:	12109:	12111:	12112:	12113:	12114:	12116:
U 31C0	12117:	12118:	12121:	12122:	12123:	12124:	12125:	12127:
U 31C8	12128:	12129:	12130:	12281:	12282:	12284:	12286:	12287:
U 31D0	12288:	12289:	12290:	12291:	12292:	12293:	12294:	12295:
U 31D8	12296:	12297:	12298:	12300:	12302:	12303:	12304:	12305:

ZZ-ENKCA-1.0
: ENKCA.MCR
:

MICRO2 1L(03) Location / Line Num 3-
3-MAR-82 09:34:33
Location / Line Number Index

C 11

3-MAR-82

ENKCA Micro-diagnostic for DAP module

Fiche 2 Frame C11

Sequence 338

Rev 01.00

Page 332

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 31E0	12306:	12307:	12309:	12310:	12311:	12312:	12313:	12314:
UU 31E8	12316:	12317:	12318:	12319:	12320:	12321:	12323:	12324:
UU 31F0	12325:	12326:	12327:	12328:	12330:	12332:	12333:	12334:
UU 31F8	12335:	12336:	12337:	12338:	12340:	12341:	12343:	12344:
UU 3200	12345:	12346:	12347:	12348:	12349:	12351:	12352:	12353:
UU 3208	12354:	12355:	12356:	12357:	12359:	12360:	12361:	12362:
UU 3210	12363:	12364:	12365:	12367:	12368:	12369:	12370:	12371:
UU 3218	12372:	12374:	12375:	12377:	12378:	12379:	12380:	12381:
UU 3220	12382:	12383:	12384:	12386:	12388:	12389:	12390:	12391:
UU 3228	12392:	12393:	12395:	12396:	12397:	12398:	12399:	12400:
UU 3230	12402:	12403:	12404:	12405:	12406:	12407:	12409:	12410:
UU 3238	12411:	12412:	12413:	12414:	12416:	12418:	12419:	12420:
UU 3240	12421:	12422:	12423:	12424:	12426:	12427:	12429:	12430:
UU 3248	12431:	12432:	12433:	12434:	12435:	12437:	12438:	12439:
UU 3250	12440:	12441:	12442:	12443:	12445:	12446:	12447:	12448:
UU 3258	12449:	12450:	12451:	12453:	12454:	12455:	12456:	12457:
UU 3260	12458:	12460:	12461:	12463:	12464:	12465:	12466:	12467:
UU 3268	12468:	12469:	12471:	12473:	12474:	12475:	12476:	12477:
UU 3270	12478:	12480:	12481:	12482:	12483:	12484:	12485:	12487:
UU 3278	12488:	12489:	12490:	12491:	12492:	12494:	12495:	12496:
UU 3280	12497:	12498:	12499:	12501:	12503:	12504:	12505:	12506:
UU 3288	12507:	12508:	12509:	12511:	12512:	12514:	12515:	12516:
UU 3290	12517:	12518:	12519:	12520:	12522:	12523:	12524:	12525:
UU 3298	12526:	12527:	12528:	12530:	12531:	12532:	12533:	12534:
UU 32A0	12535:	12536:	12538:	12539:	12540:	12541:	12542:	12543:
UU 32A8	12545:	12546:	12548:	12549:	12550:	12551:	12552:	12747:
UU 32B0	12748:	12750:	12752:	12753:	12754:	12755:	12756:	12757:
UU 32B8	12758:	12759:	12760:	12761:	12762:	12764:	12765:	12766:
UU 32C0	12767:	12768:	12770:	12771:	12772:	12773:	12774:	12776:
UU 32C8	12777:	12778:	12780:	12781:	12782:	12783:	12784:	12785:
UU 32D0	12786:	12788:	12789:	12790:	12791:	12792:	12794:	12795:
UU 32D8	12796:	12797:	12798:	12800:	12801:	12802:	12804:	12805:
UU 32E0	12806:	12807:	12808:	12809:	12811:	12812:	12813:	12814:
UU 32E8	12815:	12817:	12818:	12819:	12820:	12821:	12823:	12824:
UU 32F0	12825:	12827:	12828:	12829:	12830:	12831:	12833:	12834:
UU 32F8	12835:	12836:	12838:	12839:	12840:	12841:	12842:	12844:
UU 3300	12845:	12846:	12847:	12849:	12850:	12851:	12852:	12853:
UU 3308	12854:	12855:	12856:	12858:	12859:	12860:	12861:	12862:
UU 3310	12863:	12864:	12865:	12867:	12868:	12869:	12870:	12871:
UU 3318	12872:	12873:	12874:	12876:	12877:	12878:	12879:	12881:
UU 3320	12882:	12883:	12884:	12885:	12886:	12888:	12889:	12890:
UU 3328	12891:	12892:	12893:	12894:	12895:	12896:	12897:	12898:
UU 3330	12899:	12900:	12901:	12902:	12903:	12904:	12905:	12906:
UU 3338	12907:	12909:	12910:	12911:	12912:	12913:	12914:	12915:
UU 3340	12916:	12917:	12919:	12920:	12921:	12922:	12923:	12924:
UU 3348	12925:	12926:	12927:	12928:	12929:	12930:	12931:	12932:
UU 3350	12933:	12934:	12935:	13031:	13032:	13034:	13036:	13037:
UU 3358	13038:	13039:	13040:	13041:	13042:	13043:	13044:	13045:
UU 3360	13046:	13047:	13049:	13050:	13051:	13052:	13053:	13054:
UU 3368	13056:	13057:	13058:	13059:	13060:	13061:	13062:	13064:
UU 3370	13065:	13066:	13067:	13068:	13069:	13071:	13072:	13073:
UU 3378	13074:	13075:	13076:	13077:	13079:	13080:	13081:	13082:
U 3380	13083:	13084:	13085:	13086:	13087:	13088:	13089:	13090:

ZZ-ENKCA-1.0
: ENKCA.MCR
:

Location / Line Num 3-MAR-82
MICRO2 1L(03) 3-MAR-82 09:34:33
Location / Line Number Index

D 11

Fiche 2 Frame D11
ENKCA Micro-diagnostic for DAP module

Sequence 339
Rev 01.00

Page 333

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 3388	13091:	13092:	13093:	13094:	13095:	13096:	13097:	13098:
U 3390	13099:	13101:	13102:	13103:	13104:	13105:	13106:	13107:
U 3398	13108:	13109:	13110:	13111:	13112:	13113:	13114:	13115:
U 33A0	13116:	13117:	13118:	13119:	13120:	13191:	13192:	13194:
U 33A8	13196:	13197:	13198:	13199:	13200:	13201:	13202:	13203:
U 33B0	13204:	13205:	13206:	13207:	13208:	13211:	13213:	13214:
U 33B8	13215:	13216:	13217:	13218:	13219:	13220:	13222:	13223:
U 33C0	13224:	13225:	13227:	13228:	13229:	13230:	13232:	13233:
U 33C8	13234:	13235:	13236:	13238:	13239:	13240:	13241:	13242:
U 33D0	13243:	13244:	13245:	13246:	13247:	13248:	13249:	13251:
U 33D8	13252:	13253:	13254:	13256:	13257:	13258:	13259:	13261:
U 33E0	13262:	13263:	13264:	13265:	13266:	13268:	13269:	13270:
U 33E8	13271:	13272:	13273:	13274:	13275:	13276:	13277:	13278:
U 33F0	13279:	13281:	13282:	13283:	13284:	13286:	13287:	13288:
U 33F8	13289:	13291:	13292:	13293:	13294:	13295:	13296:	13298:
U 3400	13299:	13300:	13301:	13302:	13303:	13304:	13305:	13306:
U 3408	13307:	13308:	13309:	13310:	13312:	13313:	13314:	13315:
U 3410	13317:	13318:	13319:	13320:	13322:	13323:	13324:	13325:
U 3418	13326:	13418:	13419:	13421:	13423:	13424:	13425:	13426:
U 3420	13427:	13428:	13429:	13430:	13433:	13434:	13435:	13436:
U 3428	13438:	13439:	13440:	13441:	13442:	13444:	13445:	13446:
U 3430	13447:	13448:	13450:	13451:	13452:	13453:	13454:	13456:
U 3438	13457:	13458:	13459:	13460:	13462:	13463:	13464:	13465:
U 3440	13466:	13468:	13469:	13470:	13471:	13472:	13474:	13475:
U 3448	13476:	13477:	13591:	13592:	13594:	13596:	13597:	13598:
U 3450	13599:	13600:	13602:	13603:	13604:	13605:	13606:	13607:
U 3458	13608:	13609:	13610:	13611:	13612:	13614:	13615:	13616:
U 3460	13617:	13618:	13619:	13620:	13621:	13622:	13624:	13625:
U 3468	13626:	13627:	13628:	13630:	13631:	13632:	13633:	13634:
U 3470	13635:	13636:	13637:	13638:	13640:	13641:	13642:	13643:
U 3478	13645:	13646:	13647:	13648:	13649:	13650:	13651:	13652:
U 3480	13653:	13655:	13656:	13657:	13658:	13659:	13661:	13662:
U 3488	13663:	13664:	13665:	13666:	13667:	13668:	13669:	13670:
U 3490	13672:	13673:	13674:	13675:	13676:	13677:	13678:	13679:
U 3498	13681:	13682:	13683:	13684:	13685:	13686:	13837:	13838:
U 34A0	13840:	13842:	13843:	13844:	13845:	13846:	13847:	13848:
U 34A8	13849:	13850:	13851:	13853:	13854:	13855:	13856:	13857:
U 34B0	13858:	13859:	13861:	13863:	13865:	13866:	13867:	13868:
U 34B8	13869:	13870:	13871:	13872:	13874:	13876:	13878:	13879:
U 34C0	13880:	13881:	13882:	13883:	13884:	13885:	13886:	13888:
U 34C8	13890:	13892:	13893:	13894:	13895:	13896:	13897:	13898:
U 34D0	13899:	13901:	13903:	13905:	13906:	13907:	13908:	13909:
U 34D8	13910:	13911:	13913:	13915:	13917:	13918:	13919:	13920:
U 34E0	13921:	13922:	13923:	13924:	13926:	13928:	13930:	13931:
U 34E8	13932:	13933:	13934:	13935:	13936:	13937:	13938:	13940:
U 34F0	13942:	13944:	13945:	13946:	13947:	13948:	13949:	13950:
U 34F8	13951:	13953:	13955:	13957:	13958:	13959:	13960:	13961:
U 3500	13962:	13963:	13965:	13967:	13969:	13970:	13971:	13972:
U 3508	13973:	13974:	13975:	13976:	13978:	13980:	13982:	13983:
U 3510	13984:	13985:	13986:	13987:	13988:	13989:	13990:	13992:
U 3518	13994:	13996:	13997:	13998:	13999:	14000:	14001:	14002:
U 3520	14003:	14005:	14007:	14009:	14010:	14011:	14012:	14013:
U 3528	14014:	14015:	14016:	14018:	14020:	14022:	14023:	14024:

ZZ-ENKCA-1.0
: ENKCA.MCR
:

Location / Line Num 3-
MICRO2 1L(03) 3-MAR-82 09:34:33
Location / Line Number Index

E 11

Fiche 2 Frame E11
ENKCA Micro-diagnostic for DAP module

Sequence 340
Rev 01.00

Page 334

:Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 3530	14025:	14026:	14027:	14028:	14029:	14031:	14104:	14105:
U 3538	14107:	14109:	14110:	14111:	14112:	14113:	14114:	14115:
U 3540	14116:	14117:	14118:	14120:	14121:	14122:	14123:	14125:
U 3548	14126:	14128:	14130:	14132:	14133:	14134:	14135:	14136:
U 3550	14138:	14140:	14142:	14143:	14144:	14145:	14146:	14148:
U 3558	14150:	14152:	14153:	14154:	14155:	14156:	14158:	14160:
U 3560	14162:	14163:	14164:	14165:	14167:	14168:	14170:	14172:
U 3568	14174:	14175:	14176:	14177:	14178:	14180:	14182:	14184:
U 3570	14185:	14186:	14187:	14188:	14190:	14192:	14194:	14195:
U 3578	14196:	14197:	14198:	14200:	14202:	14204:	14205:	14206:
U 3580	14207:	14209:	14210:	14212:	14214:	14216:	14217:	14218:
U 3588	14219:	14220:	14222:	14224:	14226:	14227:	14228:	14229:
U 3590	14230:	14232:	14234:	14236:	14237:	14238:	14239:	14240:
U 3598	14242:	14372:	14373:	14375:	14377:	14378:	14379:	14380:
U 35A0	14381:	14382:	14383:	14384:	14385:	14386:	14388:	14389:
U 35A8	14390:	14391:	14392:	14393:	14394:	14396:	14398:	14400:
U 35B0	14401:	14402:	14403:	14404:	14405:	14406:	14407:	14409:
U 35B8	14411:	14413:	14414:	14415:	14416:	14417:	14418:	14419:
U 35C0	14420:	14422:	14424:	14426:	14427:	14428:	14429:	14430:
U 35C8	14431:	14432:	14433:	14435:	14437:	14438:	14439:	14440:
U 35D0	14441:	14443:	14444:	14445:	14447:	14448:	14449:	14450:
U 35D8	14451:	14452:	14454:	14456:	14458:	14459:	14460:	14461:
U 35E0	14462:	14464:	14466:	14467:	14468:	14469:	14470:	14472:
U 35E8	14473:	14474:	14476:	14477:	14478:	14479:	14480:	14482:
U 35F0	14484:	14486:	14487:	14488:	14489:	14490:	14491:	14493:
U 35F8	14495:	14496:	14497:	14499:	14693:	14694:	14696:	14698:
U 3600	14699:	14700:	14701:	14702:	14703:	14704:	14705:	14706:
U 3608	14707:	14708:	14709:	14711:	14712:	14713:	14714:	14715:
U 3610	14716:	14718:	14720:	14722:	14723:	14724:	14725:	14726:
U 3618	14728:	14730:	14732:	14733:	14734:	14735:	14736:	14738:
U 3620	14740:	14742:	14743:	14744:	14745:	14746:	14747:	14749:
U 3628	14751:	14752:	14753:	14755:	14756:	14757:	14758:	14759:
U 3630	14761:	14763:	14765:	14766:	14767:	14768:	14769:	14770:
U 3638	14772:	14774:	14776:	14777:	14778:	14779:	14780:	14781:
U 3640	14783:	14785:	14787:	14788:	14789:	14790:	14791:	14793:
U 3648	14795:	14796:	14798:	14799:	14800:	14801:	14802:	14803:
U 3650	14805:	14807:	14809:	14810:	14811:	14812:	14813:	14814:
U 3658	14816:	14818:	14819:	14820:	14822:	14823:	14824:	14825:
U 3660	14826:	14827:	14829:	14831:	14833:	14834:	14835:	14836:
U 3668	14837:	14838:	14840:	14842:	14843:	14845:	14846:	14847:
U 3670	14848:	14849:	14850:	14852:	14854:	14856:	14857:	14858:
U 3678	14859:	14860:	14861:	14863:	14865:	14866:	14867:	14868:
U 3680	14870:	14871:	14872:	14873:	14874:	14875:	14877:	14879:
U 3688	14881:	14882:	14883:	14884:	14885:	14886:	14888:	14890:
U 3690	14891:	14892:	14893:	14895:	14896:	14897:	14898:	14899:
U 3698	14900:	14902:	14904:	14906:	14907:	14908:	14909:	14910:
U 36A0	14911:	14913:	14915:	14916:	14917:	14918:	14919:	14921:
U 36A8	14922:	14923:	14925:	14926:	14928:	14929:	14930:	14931:
U 36B0	14932:	14933:	14934:	14935:	14937:	14939:	14941:	14942:
U 36B8	14943:	14944:	14945:	14946:	14948:	14950:	14951:	14952:
U 36C0	14954:	14958:	14959:	14962:	14964:			

ENKCA Words not
in bounds 512

Used 512
Remaining

Total microwords used in memory C: 512
Total microwords remaining in memory C: 0
Highest address used in memory C: 180 (hex)

ZZ-ENKCA-1.0 ;
: ENKCA.MCR
:

MICRO2 1L(03) U-code Microword Su 3-MAR-82
U-code Microword Summary

G 11
Fiche 2 Frame G11
ENKCA Micro-diagnostic for DAP module
Sequence 342
Rev 01.00
Page 336

ENKCA Words not
in bounds 5061
Used 5061
Remaining

Total microwords used in memory U: 5061
Total microwords remaining in memory U: 0
Highest address used in memory U: 36C4 (hex)

ZZ-ENKCA-1.0 :
: ENKCA.MCR
:
MICRO2 1L(03) Error Summary
Error Summary

H 11
3-MAR-82
Fiche 2 Frame H11
ENKCA Micro-diagnostic for DAP module
Sequence 343
Rev 01.00
Page 337

Pass 1 warnings: 0
Pass 1 errors: 0

Pass 2 warnings: 0
Pass 2 errors: 0